UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Desenvolupament d'una llibreria i d'un editor gràfic d'Interfícies d'Usuari per a videojocs

– Treball Final de Grau –

Grau en Disseny i Desenvolupament de Videojocs

Autor: Garrigó Invers, Marc
Pla d'estudis 2014

Director: Díaz Iriberri, Jose

Resum

L'objectiu d'aquest projecte final de grau és el de desenvolupar un programa a través del qual es puguin dissenyar i implementar interfícies d'usuari per a videojocs. Gairebé totes les aplicacions digitals necessiten una interfície d'usuari, i els videojocs no en són l'excepció. Tot i que ja existeixen alguns programes per a crear interfícies d'usuari, no n'hi ha cap específic per a videojocs, i aquí és on entra aquest projecte. Donat que un programa d'aquest tipus pot arribar portar algun any de desenvolupament amb un equip gran de persones darrere, aquest projecte en concret es tracta d'un prototip, per a analitzar la viabilitat de mantenir-ne el seu desenvolupament al llarg del temps.

El desenvolupament d'aquest programa es fa a través de Visual Studio, en c++. Són necessàries dues parts, una llibreria d'interfície d'usuari, que gestiona tota la lògica dels elements que es mostren per pantalla i una segona part, un editor gràfic que permet dissenyar la interfície fàcilment. Tot i tractar-se d'un projecte altament tècnic, s'explica tot el procediment de manera que una persona sense coneixement de programació pugui seguir com s'han dut a terme les diferents tasques i, a la vegada, que una persona amb coneixements tècnics que llegeixi aquest document i vulgui realitzar una implementació semblant, sigui capaç d'entendre com s'ha fet.

En acabar aquest projecte s'ha arribat a la conclusió que seria interessant seguir desenvolupant aquest programa i donar-li visibilitat, per tal que aquells qui el trobin útil el puguin utilitzar i, si es dóna l'ocasió, un equip de persones es pugui encarregar de portar el programa a un nivell més elevat i comercialitzar-lo.

Per a aquells interessats en el projecte, es pot accedir a la pàgina web de Github per veure'n el codi font i a l'apartat de "release" per a baixar-se l'ultima versió.

Pàgina web de Github: https://github.com/markitus18/ThorUI

Apartat de releases: https://github.com/markitus18/ThorUI/releases

Paraules clau

UI, interfície d'usuari, programació, editor gràfic, videojocs, C++

Índex

Índex de figures

Índex de taules

1. Introducció

1.1 Motivació

La motivació principal d'aquest projecte és la de produir una eina que pugui ser usada per desenvolupadors de videojocs per a dissenyar i implementar la interfície d'usuari (UI: User Interface), de manera que ajudi en el procés de desenvolupament d'un videojoc i estalviï temps a l'equip que el porti a terme.

Tenint experiència prèvia en haver realitzat un petit sistema d'interfície d'usuari, el projecte semblava una oportunitat perfecte per a portar aquests coneixements al següent nivell. La UI és un sistema que s'utilitza en gairebé tots els videojocs actuals i, en equips de mida mitjana, es pot trobar un departament sencer dedicat a dissenyar-la i implementar-la.

Val la pena afegir que desenvolupar un programa com aquest requereix d'un codi molt ben estructurat i organitzat sent, així, un gran repte per proposar-se. Un treball d'aquest tipus servirà per posar a prova els coneixements previs en programació i, a la vegada, ampliar-los.

1.2 El problema

El desenvolupament d'un videojoc requereix de la unió de moltes disciplines diferents, de molt temps i paciència. Actualment hi ha una gran quantitat de programes enfocats per a millorar aquest procés.

Un dels sistemes que formen part de gairebé qualsevol videojoc és la interfície d'usuari. Desenvolupar-la requereix d'un sistema força complex i ben estructurat i, per tant, requereix de temps. Sovint aquest sistema acaba portant a problemes de codi atès que interactua amb la resta de sistemes del videojoc.

Un programa que faciliti el desenvolupament de la UI pot tenir una bona rebuda en el sector, ja que l'equip encarregat d'aquesta part del videojoc només s'hauria de preocupar pel disseny i no tant per la implementació. Proporcionar una estructura compacta del sistema evitaria molts problemes de desenvolupament a llarg termini.

1.3 Conceptes bàsics

Per a poder comprendre correctament tots els aspectes tractats en aquest projecte cal prèviament entendre un conjunt de conceptes bàsics de programació. Per a aquells que ja tinguin coneixements de programació aquest apartat no és necessari llegir-lo.

- Memòria d'un programa: es pot entendre com una espècie de magatzem dins de l'ordinador on un programa pot guardar la informació que necessiti.

```
int numero = 1; //Varaible de número amb valor 1
char text = 'c'; //Variable de caràcter amb valor 'c'
```

[Fig. 1-1] Exemple de variables

- Variables: part de la memòria en la que es guarda un valor. En la Figura 1-1 es poden veure dos exemples de variables.

- Funció: conjunt de línies de codi que realitzen alguna tasca individual . Ja sigui des d'una simple suma de valors o alguna acció més complexa com el moviment d'un personatge. En la Figura 1-2 se'n pot veure un exemple.

```
void funcio() //nom de la funcio
{
    //accions de la funcio
    //pot tenir multiples linies
}
```

[Fig. 1-2] Exemple de funció

- Classes i objectes: una classe és un tipus de dades definit per l'usuari. Combinen la representació de dades i funcions per a manipular aquestes dades. Es poden entendre com la definició d'un element en el món. Quan es parla de què és, per exemple, una taula i els trets característics que defineixen. El conjunt d'aquestes característiques formaria una classe.

  Un objecte, per altra banda, és una instància d'una classe concreta. Mentre que les classes defineixen un tipus de dades, per no existeixen enlloc, l'objecte porta aquest conjunt de dades a existir en una zona de la memòria.

En un programa es pot tenir una classe anomenada *Taula*, en la qual s'hi troben les característiques que comparteixen totes les taules, i s'hi poden trobar múltiples objectes que pertanyen a aquesta classe, amb petites diferències entre ells. A l'esquerra (Fig. 1-3) es pot veure l'exemple d'una classe, que conté una funció i dues variables. A la dreta (Fig. 1-4), tres objectes pertanyents a la classe de l'esquerra.

```
class Nom_Classe
{
    void funcio()
    {

    }

    int variable_1;
    char variable_2;
};
```

[Fig. 1-3] Exemple de classe

```
Nom_Classe objecte1;
Nom_Classe objecte2;
Nom_Classe objecte3;
```

[Fig. 1-4] Exemple d'objectes

- Llibreria: en l'àmbit de la programació, una llibreria és un conjunt de fitxers de codi que defineixen unes funcions i unes classes de cara a facilitar la programació a alguna altra persona. Es poden trobar llibreries de tot tipus, des de càlculs matemàtics fins a comportaments d'intel·ligència artificial. Cal diferenciar una eina (o programa) de UI d'una llibreria. Mentre un programa permet dissenyar una interfície de manera gràfica, a través de botons a la pantalla, una llibreria és únicament codi que s'implementa dins d'un altre programa.

- Frame: per entendre què és un frame, cal primer entendre que un videojoc (i, en general, qualsevol programa) s'executa en un bucle. El codi es va executant contínuament, una i una altra vegada, fins al final del programa. Un frame, doncs, és una iteració, una etapa d'aquest bucle.

- Widget (Window Gadget): pel que fa a interfícies d'usuari s'anomena widget a aquell element que apareix a la pantalla i que forma part de la interfície, com per exemple un botó, una imatge o una barra de desplaçament (*scrollbar* en anglès)

Per a una explicació més àmplia sobre les bases de C++, en la web B3 es troben un conjunt de tutorials en què s'expliquen tots els conceptes mencionats anteriorment.

1.4 Objectius generals

L'objectiu principal és aconseguir la major quantitat de coneixements relacionats amb la planificació i la gestió d'un projecte i el desenvolupament d'aquest en concret on s'aplicaran, majorment, conceptes de programació apresos durant el grau.

- Aprendre a portar la gestió d'un projecte en la seva totalitat.
- Aplicar i posar a prova els coneixements adquirits durant el grau.
- Adquirir més experiència en quant a la gestió del temps, la planificació de les tasques i l'estimació d'hores.
- Aprendre a solucionar imprevistos de manera efectiva.
- Millorar en la recerca d'informació: adquirir més recursos per a buscar informació necessitada i discernir la informació correcta de l'enganyosa.
- Saber posar-se en el punt de vista de l'usuari d'una eina i desenvolupar funcionalitats útils i intuïtives per a aquest.
- Aprendre a desenvolupar una llibreria i saber implementar-la en un programa.

1.5 Objectius específics

L'objectiu especific d'aquest projecte es centra en aconseguir un prototip d'un programa que es pugui utilitzar durant el desenvolupament d'un videojoc per dissenyar la interfície d'usuari de la manera més senzilla possible. Tenint en ment que el desenvolupament d'un programa acabat d'aquest tipus pot portar anys, l'objectiu del treball és la realització d'un prototip per a demostrar que la base funciona i es pot utilitzar, de manera que es pugui ampliar de cara al futur amb noves funcionalitats.

Per tal de tenir aquest prototip, és necessari el desenvolupament d'una llibreria que faciliti la gestió i la creació de la interfície d'usuari i, per sobre d'aquesta llibreria, una editor per a que es pugui dissenyar la UI de manera gràfica, sense tenir la necessitat de programar.

1.6 Abast del projecte

Pel que fa a l'abast del projecte, cal tenir molt clara la seva mida. Un programa d'aquest tipus podria portar uns anys de desenvolupament amb un equip gran de persones. Tenint en compte que els recursos d'aquest projecte són limitats, és important marcar-ne els seus límits per assegurar-ne l'èxit. No es tractarà d'un programa que es pugui utilitzar per a dissenyar i implementar qualsevol interfície d'usuari, sinó d'un prototip, en el qual es podran generar interfícies bàsiques per a un videojoc. Així doncs, per poder assolir aquest repte, caldrà definir quines tasques són necessàries per tenir el nucli del programa acabat, i no desviar els esforços en tasques que, tot i ser útils, no formin part estrictament d'aquest nucli.

Aquest projecte va dirigit principalment a estudis petits de videojocs que no necessitin d'una interfície molt complexa. A través d'aquest programa, podran estalviar temps i esforços i dedicar-los a altres parts importants en el seu desenvolupament. D'altra banda, al llarg del grau en videojocs s'han anat desenvolupant diversos projectes amb interfícies senzilles i una eina d'aquest tipus podria ajudar a millorar-ne la qualitat dels jocs, així

que també seria útil per a estudiants que estan començant a desenvolupar els seus primers videojocs.

Finalment, hi haurà diversos perfils que es beneficiaran d'aquest projecte. Primerament els mencionats abans, que utilitzaran directament l'eina. Després, els jugadors d'aquells videojocs que s'hagin desenvolupat utilitzant aquest programa ja que, si l'equip disposa de més temps, els videojocs comptaran amb més qualitat. Per acabar, aquest projecte beneficiarà també a aquells interessats en el desenvolupament d'eines d'aquest tipus, ja que podran aprendre'n dels aspectes bons i dolents que hi hagi hagut durant el desenvolupament i aplicar aquests coneixements en les seves pròpies eines.

2. Context

En aquest apartat s'exposa el context necessari per entendre degudament l'anàlisi de l'estat de l'art.

Segons l'estructura interna del codi d'una llibreria d'interfície d'usuari (UI: User Interface) es poden classificar en dos tipus: llibreries de mode retingut (o de *framework*) i llibreries de mode immediat. A continuació s'explica en què consisteix cada tipus i quins són els seus avantatges i inconvenients.

2.1 Llibreries de mode retingut (RMGUI)

Aquest tipus de llibreria centra la seva estructura en l'ús de classes. La llibreria proporciona un conjunt de classes per tal que la persona que implementi la UI les utilitzi. Aquestes classes solen ser elements que es poden identificar a la pantalla fàcilment: un botó, una imatge, un text o un menú, entre d'altres. La llibreria gestiona tota la part lògica d'aquests elements, mentre que la persona encarregada d'implementar-la els col·loca per la pantalla segons requereixi el disseny. També programa quines són les accions que passen quan l'usuari de l'aplicació interactua amb la interfície.

Avantatges:

- Les funcionalitats que porta es solen poder utilitzar fora del marc de UI. Per exemple, càlculs de matrius que es poden utilitzar en algun altra part del codi.
- Permet optimitzar la part del render (mostrar els elements a la pantalla) ja que, al tenir tots els objectes creats, es poden agrupar per pintar-los a la vegada.
- Resulta més fàcil separar el codi pertanyent a la UI respecte al d'altres funcionalitats.

- Es pot canviar l'aparença predefinida dels objectes més fàcilment, ja que es tenen les dades guardades a memòria.
- La llibreria pot decidir no tornar a pintar la pantalla a cada frame, ja que és més fàcil detectar quan no hi ha hagut canvis, com de posició o de color i, per tant, utilitzar l'última imatge generada.
- Es més fàcil serialitzar la UI, es a dir, guardar tota la informació dels elements en un fitxer.

Inconvenients:

- El procés d'aprenentatge per a qui ha d'utilitzar aquesta eina és més llarg atès que, normalment, requereix de més coneixement.
- Aquest tipus de llibreria sol ser gran, amb molta quantitat de línies de codi i fitxers i, per tant, necessita d'equips més grans per desenvolupar-la i mantenir-la.
- El temps de càrrega de les escenes és més llarg, ja que es necessita crear els objectes que apareixeran per pantalla i, a més, ocupa més memòria.

2.2 Llibreries de mode immediat (IMGUI)

Al contrari d'una llibreria de mode retingut, aquesta es basa en l'ús de funcions. En comptes de crear un objecte «Botó», s'utilitza una funció per especificar que es vol un botó (o qualsevol altre element) en algun lloc de la pantalla. Igual que l'altre tipus de llibreria, cal programar quins són els resultats d'interactuar amb la interfície.

Avantatges:

- És més fàcil crear elements personalitzats, es a dir, nous elements que no formin part de la llibreria i que es necessitin per a un projecte concret.
- L'extensió en quant a fitxers i línies de codi és molt més petita respecte a la llibreria de mode retingut.
- Donat que la llibreria és més petita, es necessita un equip més petit. A més, si el propietari de la llibreria canvia, una altra persona pot agafar el rol més fàcilment.
- No necessita gestionar els objectes a memòria ja que no n'hi ha.

Inconvenients:

- S'ha de tornar a renderitzar la pantalla a cada frame. Per a certs programes no és un inconvenient però, en el cas dels videojocs, és molt necessari estalviar tant temps com sigui possible al executar el codi, ja que el rendiment del joc es pot veure afectat si no es té cura.
- No és fàcil desenvolupar un sistema en el qual gent externa contribueixi a la llibreria implementant nous widgets.
- No hi ha un sistema natiu d'internacionalització, es a dir, gestionar diferents llenguatges dins del joc. En el cas que es necessités aquest sistema, cosa que en la majoria de videojocs és així, la persona que programa la UI l'hauria d'implementar.
- La gestió dels widgets en canviar la mida de la pantalla és més complicada de personalitzar i d'aquesta manera es troba lligat al sistema que hi hagi implementat.
- Resulta més complicat modificar l'aparença dels objectes, ja que és difícil especificar com ha de ser un element només utilitzant una funció.
- Widgets més avançats, que no es trobin en la llibreria, com podria ser una barra de progrés, són més rebuscats de programar que en l'altre tipus de llibreria.

Una vegada llistats tots els avantatges i inconvenients de cada tipus, es pot fer una comparació entre ells, tractant els aspectes més importants.

Un dels aspectes primordials en els videojocs és l'optimització del temps que triga en executar-se el codi ja que, com pitjor sigui el rendiment, més lent s'executa el videojoc. Com s'ha vist anteriorment, pel que fa a velocitat, les llibreries RMGUI són més lentes a l'inici d'una escena (al carregar-la) però són més ràpides en la resta de casos. Si bé en un videojoc no és crític fer esperar al jugador algun segon més en carregar una escena, és imprescindible que no baixi el rendiment mentre el jugador realitza alguna acció, ja que això comporta una frustració per a ell. Les llibreries de UI de mode retingut, per tant, tenen un gran factor a tenir en compte quant a optimització.

Pel que fa a la quantitat de codi de cada llibreria, és molt clar que les de tipus IMGUI són molt més petites. El fet de tenir menys codi implica que el fitxer del programa és més petit i també estalvia temps durant el desenvolupament del videojoc. Aquest punt seria molt interessant de cara a les llibreries IMGUI si no fos perquè tota la simplicitat que comporta la llibreria es perd quan el programador que l'utilitza ha d'implementar la interfície. Un dels inconvenients de les llibreries IMGUI és que no és fàcil programar widgets personalitzats ni canviar-los-hi l'aparença. Aquesta manca fa que, per a poder fer una UI complexa de la mateixa manera que es faria amb una llibreria RMGUI, el programador hagi d'escriure molt més codi i, al final, la quantitat de codi que s'utilitza en ambdues llibreries es semblant. Per tant, respecte a la quantitat de codi, totes dues llibreries són equiparables, però en la llibreria RMGUI la complexitat recau en el codi de la pròpia llibreria i, per tant, estalvia temps a qui la utilitza.

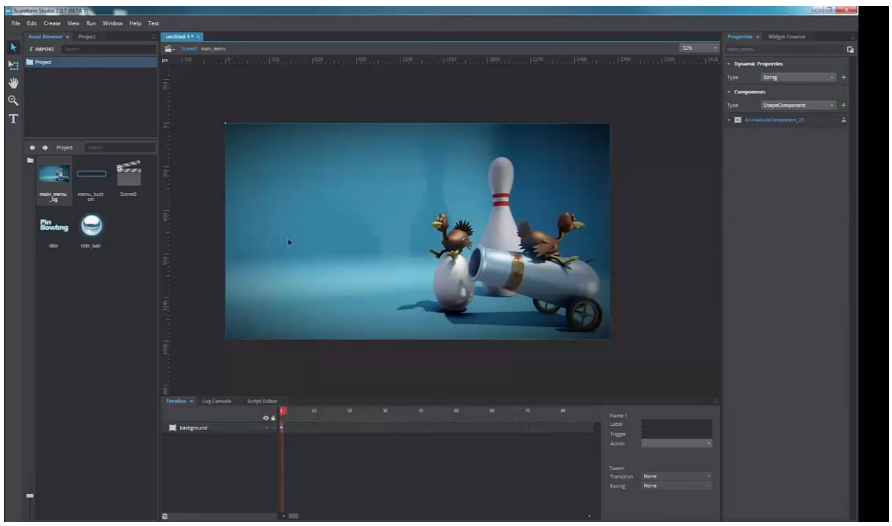
Finalment, una mica relacionat amb el punt anterior, falta parlar del temps d'implementació d'una i altra llibreria. Si bé a través de una llibreria IMGUI es pot implementar una UI senzilla en molt poc temps, quan la UI comença a ser més complexa, la quantitat de temps invertit augmenta exponencialment. En canvi, en una llibreria de tipus RMGUI es triga una mica més en generar una interfície senzilla, però l'estalvi de temps en interfícies complexes és destacable.

3. Estat de l'art

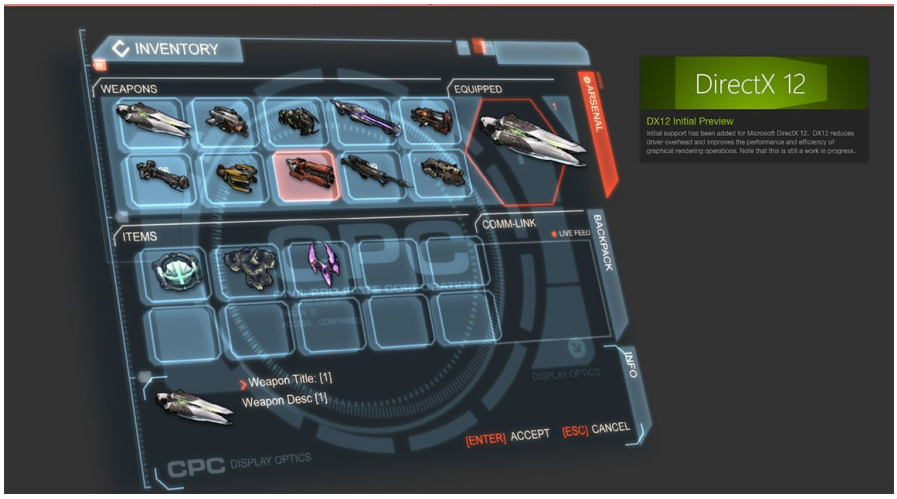
En aquest apartat es fa un estudi dels diferents programes i llibreries d'interfície d'usuari (UI: User Interface) que es poden trobar, ja siguin gratuïtament o de pagament. Donat que en l'apartat de context ja s'ha comparat extensament els dos tipus de llibreries de UI que existeixen, en aquest es centra en els punts específics a destacar de cadascuna.

3.1 Scaleform

Scaleform és el programa de major referència per a aquest treball. És un programa que utilitza una llibreria de mode retingut, desenvolupat per Autodesk i integrat en un projecte més gran, Stingray, un motor de joc. A la Fig. 3-1 es pot veure la interfície d'aquest programa i a la següent imatge (Fig. 3-2), un exemple d'interfície generada amb Scaleform.



[Fig. 3-1] Scaleform Studio - Autodesk



[Fig. 3-2] Exemple de UI de Scaleform

Aquest programa ofereix la gran majoria de funcionalitats que es necessiten per a poder dissenyar i implementar una interfície d'usuari en un videojoc. Té una part gràfica molt ben tractada, que permet personalitzar la interfície com es necessiti per al projecte, així com una gran optimització del rendiment, ja que està pensat específicament per a videojocs. També té integrat un sistema d'àudio i un gran equip darrere per donar suport al programa, desenvolupar-lo segons les necessitats actuals i resoldre dubtes als clients.

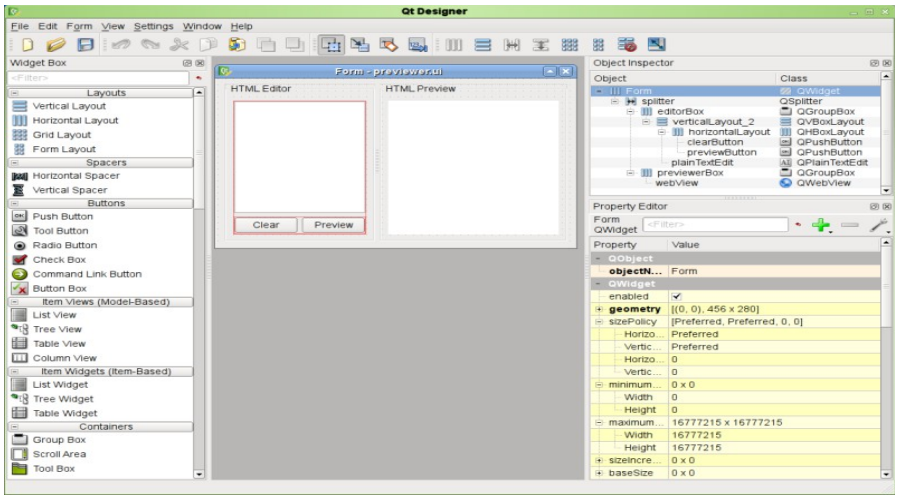
Tot i semblar el programa ideal per a dissenyar interfícies d'usuari per a videojocs, té dos grans inconvenients. Primerament, és un programa d'Autodesk i, com tota la resta de programes d'aquesta empresa, és de pagament. Tot i que el preu de la llicència de Scaleform ja no es troba disponible, es poden agafar altres programes d'Autodesk com a referència, que tenen una llicència mensual d'uns 200€. Això implica que clarament el client objectiu d'aquest programa són les grans produccions de videojocs, que es poden permetre pagar aquesta llicència, ja que els petits estudis no solen tenir prou pressupost. Aquesta suposició es reforça en analitzar els videojocs que es troben al mercat que han utilitzat aquest programa, com Crysis, Skyrim o Battlefield 3, sent tots grans produccions amb amplis pressupostos.

El segon gran inconvenient de Scaleform és que, a data de 7 de gener de 2018, Autodesk ha retirat aquest programa i el seu motor de joc (Stingray) del mercat deixant, així, de desenvolupar-los i de proporcionar suport per a aquests. Es pot consultar la notícia en la web de Stingray (B4). Si bé aquesta notícia podria causar una alerta en aquest projecte, el motiu pel qual l'equip d'Autodesk ha decidit retirar-se és degut a que no poden competir contra Unity i Unreal Engine en quant a motor de joc. És una decisió que no té res a veure amb el programa de UI.

Tot i que les pàgines oficials de Scaleform ja no són operatives, en aquest vídeo (B5) es parla de manera força detalladasobre el programa.

3.2 Qt

Qt és un altre referent a tenir en compte. Es tracta d'un programa semblant a Scaleform, que consta d'un editor gràfic per tal de dissenyar la interfície i una llibreria de mode retingut. Al contrari que Scaleform, Qt està orientat a qualsevol tipus de programa, ja sigui ordinador, mòbil o qualsevol dispositiu equipat amb una pantalla. A continuació es pot veure el seu editor gràfic, Qt Designer (Fig. 3-3), i un exemple d'interfície d'una rentadora creada amb aquest programa (Fig. 3-4)



[Fig. 3-3] Qt Designer – Editor gràfic de Qt

[Fig. 3-4] Exemple de UI feta amb Qt

Un dels punts més forts de Qt és que compta amb gairebé de tot. Té una gran llista de diferents widgets que es poden implementar i un sistema per crear-ne de personalitzats de manera molt senzilla. Tot i comptar amb un editor gràfic, permet implementar la UI a través de la programació, mitjançant la seva llibreria. És un sistema tant potent que la majoria de gent acaba escollint-lo vers a fer-ho utilitzant l'editor.

Si bé la gran complexitat és un aspecte bo de Qt, ja que arriba a una gran diversitat de programes, també n'és un dels seus punts febles. Descarregar-se els fitxers bàsics necessaris per a utilitzar aquest programa requereix de 30 GB d'espai al disc dur. Utilitzar aquesta llibreria requereix d'un procés d'aprenentatge llarg. Si bé tenir els conceptes bàsics per a fer una interfície senzilla és prou assolible, conèixer totes les funcionalitats de la llibreria requereix d'una gran quantitat de temps, tot i que hi ha una bona documentació que en facilita el procés.

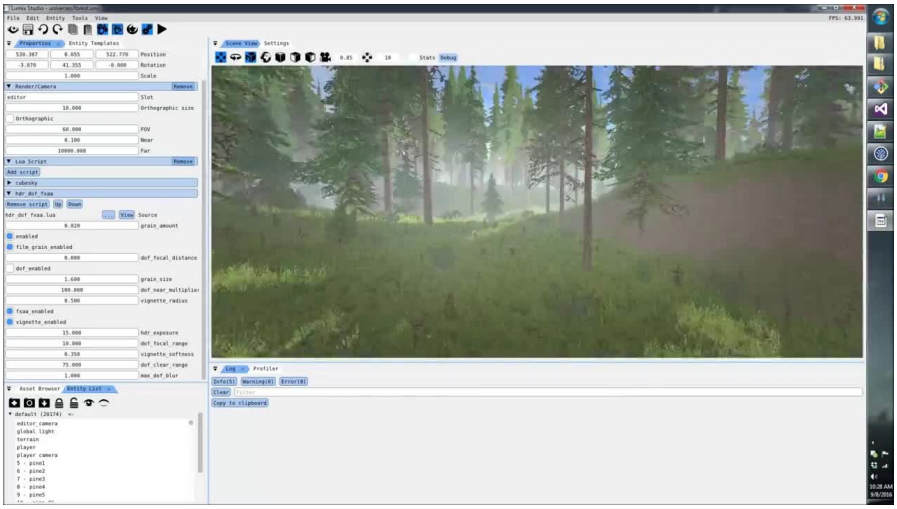
Finalment, donat que la llibreria intenta englobar la major diversitat de programes possibles i, en general, no necessiten un molt bon rendiment del codi, Qt deixa de banda la part d'optimització. D'aquesta manera, per tant, Qt no és ideal per a videojocs, sobretot els que requereixen d'un gran ús de les capacitats del dispositiu on s'executen.

Per a més informació sobre Qt en la seva pàgina oficial (B6) s'informa sobre totes les seves possibilitats.

3.3 dear-ImGui

Aquesta és una llibreria força coneguda que, com el seu nom indica, és de mode immediat (IMGUI). El gran fort d'aquesta és la seva simplicitat ja que, amb no més de quatre fitxers de codi permet crear una interfície d'usuari de manera molt ràpida amb widgets senzills. Un altre dels aspectes a tenir en compte d'aquesta llibreria és que compta amb una gran comunitat darrere que li dona suport i contribueix, tant amb codi com amb suggeriments, a que la llibreria avanci. Cal remarcar que aquesta llibreria va ser creada per una sola persona, tot i que actualment n'és una altra qui n'està al càrrec.

A continuació (Fig. 3-5) es pot veure un exemple d'un motor de joc (Lumix Engine) que utilitza dear-Imgui



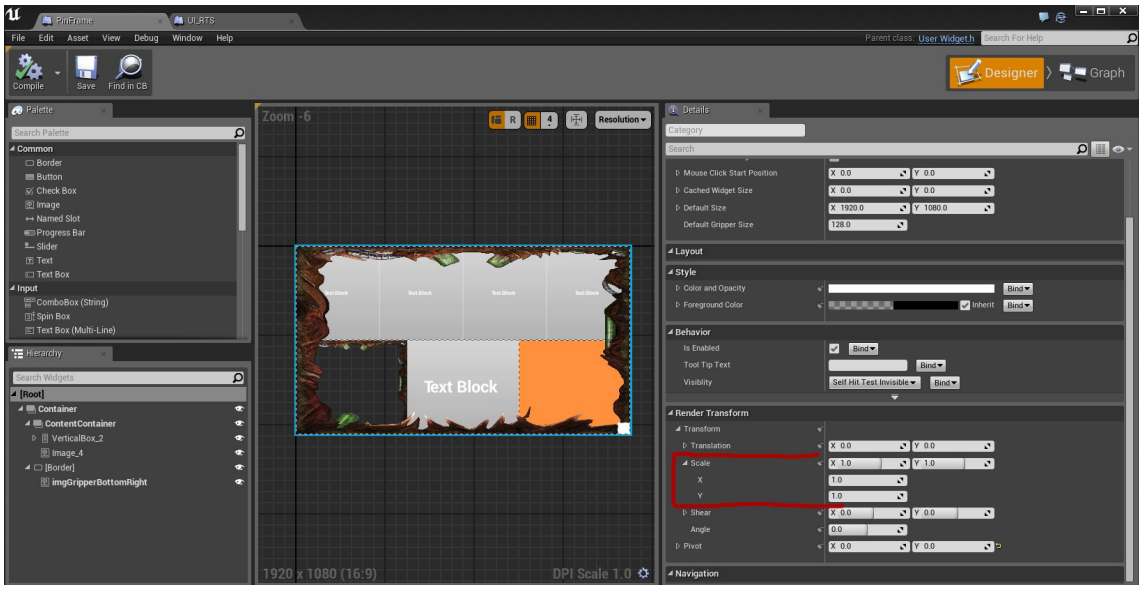
[Fig. 3-5] Lumix Engine - Exemple de UI feta amb dear-ImGui

La part negativa d'aquesta llibreria és que està molt enfocada al desenvolupament d'eines, no de jocs, amb la qual cosa dóna més importància a la funcionalitat que a la visualització de la interfície. A més a més, després de la comparació dels dos tipus de llibreries de UI, s'ha pogut veure que el mode immediat no és el millor per a videojocs.

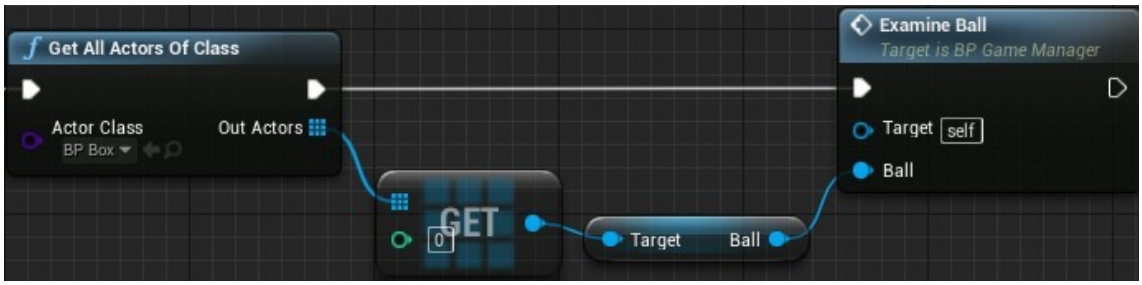
Per a més informació sobre dear-ImGui, es pot accedir al seu codi des de la pàgina de Github (B7) i a diferents extensions a la Wiki de Github (B8). Malauradament, no hi ha cap pàgina oficial de documentació de dear-ImGui,

3.4 Unreal Engine

Tot i no ocupar el mateix lloc en el sector dels videojocs que aquest projecte, Unreal Engine és un motor de joc a tenir en compte a l'hora d'avaluar un programa per dissenyar una UI, ja que compta amb una interfície gràfica per fer-ho de manera ràpida i senzilla. A més, és un programa pensat per al desenvolupament de videojocs i, per tant, té cura amb la optimització dels recursos. Pel que fa a programar les accions de la UI, Unreal té una eina molt útil, que anomena *blueprints*, a través de la qual es programa mitjançant nodes d'un gràfic. En les Figures 3-6 i 3-7 es pot veure l'editor gràfic d'Unreal i el seu sistema de *blueprints* per a programar la UI.



[Fig. 3-6] Editor gràfic d'Unreal Engine



[Fig. 3-7] Unreal Engine blueprints

Al llarg del projecte Unreal s'utilitza com a referència per a extreure coneixement a l'hora de programar la llibreria i l'editor gràfic, més que considerar-lo com a una competència. Per a més informació sobre la UI de Unreal, es pot consultar a la seva pàgina de documentació (B9).

3.5 Unity

Unity és un altre motor de joc que, juntament amb Unreal, ocupa les primeres posicions pel que fa a motors. Unity és més triat per equips "indie", es a dir, projectes amb pressupostos baixos i equips petits, però és una eina molt completa per a desenvolupar un videojoc. Tot i la seva importància en el món dels videojocs, Unity no té una gran eina per a implementar la interfície d'usuari, amb la qual cosa no és necessari entrar a analitzar-la. Unity disposa d'una pàgina web oficial (B10) per a consultar més informació i descarregar-se el motor.

3.6 Conclusions de l'anàlisi

Com s'ha pogut veure analitzant les diferents llibreries i programes d'interfície d'usuari més conegudes, no n'hi ha cap que permeti generar la interfície de manera independent al codi del programa. Si bé és cert que els motors de joc (Unity i Unreal Engine) compten amb un sistema de UI, aquells desenvolupadors que no vulguin utilitzar un motor de joc d'aquest estil per als seus videojocs no poden fer servir les seves eines. La idea principal del projecte, per tant, serà desenvolupar una eina que permeti generar aquesta interfície de manera separada al joc. Tot i que aquest sistema implicaria que els equips que utilitzessin Unity i Unreal, entre d'altres, no podrien fer servir el programa proposat en aquest projecte, es pot considerar, de cara al futur, adaptar la llibreria per a que es pugui utilitzar des d'altres motors de joc.

4. Planificació

En aquest apartat es tractarà la planificació duta a terme per desenvolupar el projecte. Amb previsió d'una gran quantitat de feina impossible de realitzar durant els quatre mesos de durada del projecte, s'ha decidit començar amb antelació. El projecte es desenvoluparà en dues parts (setembre a febrer i febrer a juliol) coincidint, així, amb els períodes lectius del calendari acadèmic.

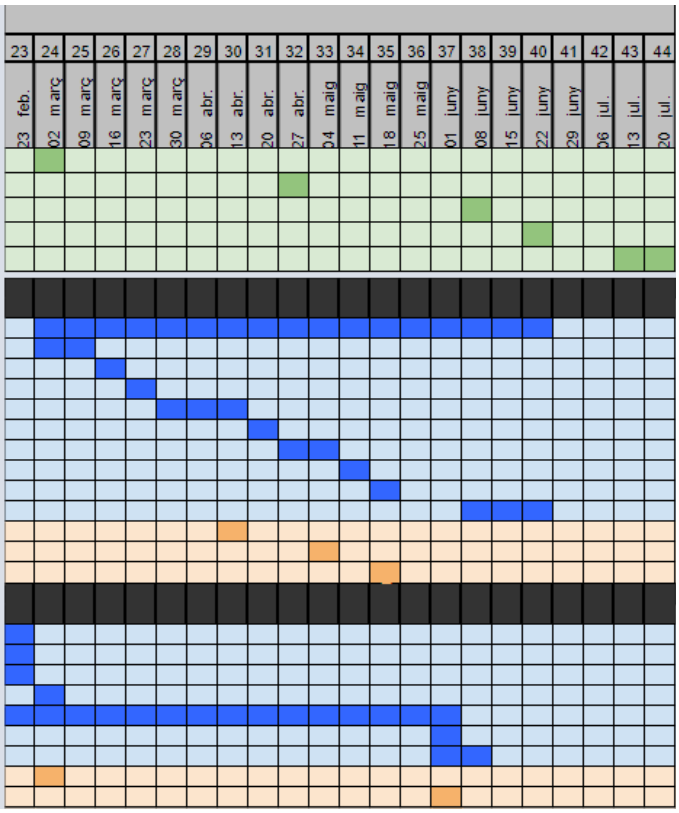
A la primera part és on es realitza, prèviament al desenvolupament del programa, una recerca de l'estat de l'art, per tal de poder veure degudament quines necessitats actuals hi ha en els videojocs i quina és la millor manera d'enfocar el projecte. En aquesta part és on s'apliquen, majorment, coneixements ja obtinguts durant el grau. En la següent taula (4-1) es pot observar el diagrama de Gantt utilitzat per a la primera part del desenvolupament del projecte.

		Duration	Beggning week	End week	Weeks, friday to friday																						
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
					22 set.	29 set.	06 oct.	13 oct.	20 oct.	27 oct.	03 nov.	10 nov.	17 nov.	24 nov.	01 des.	08 des.	15 des.	22 des.	29 des.	05 gen.	12 gen.	19 gen.	26 gen.	02 feb.	09 feb.	16 feb.	23 feb.
D1.1	Rubric 1	1	24	24																							
D1.2	Rubric 2	1	32	32																							
D1.3	Rubric 3	1	38	38																							
D1.4	TFG Delivery	1	40	40																							
D1.5	Defense	2	43	44																							
WP1	Development - Part 1	46	1	22																							
T1.1	Project	21	2	22																							
T1.2	OpenGL + SDL base	2	2	3																							
T1.3	Keyboard + mouse input	1	4	4																							
T1.4	Basic items 1: image and button	2	4	5																							
T1.5	Color system	1	6	6																							
T1.6	Fonts system	1	6	6																							
T1.7	Base items 2: text	1	7	7																							
T1.8	Basic editor	3	7	9																							
T1.9	Load/Save scenes	2	10	11																							
T1.10	Editor: dock system	2	12	13																							
T1.11	Editor: hierarchy display	1	14	14																							
T1.12	Editor: gizmos	2	15	16																							
T1.13	2D Transformations	3	17	19																							
T1.14	Editor: selecting items	1	20	20																							
T1.15	Hierarchy item control	3	20	22																							
M1.1	Basic items implemented	1	7	7																							
M1.2	Editor with scene saves	1	10	10																							
M1.3	Gizmos + transformations	1	19	19																							
M1.4	Item pareting control	1	22	22																							
WP2	Documentation - Part 1	25	1	25																							
T1.1	State of the art	1	1	1																							
T1.2	Planning	1	1	1																							
T1.3	Methodology	1	1	1																							
T1.4	Development	21	2	22																							
M1.1	Pre-Project research	1	1	1																							

[Taula 4-1] Diagrama de Gantt - Primera part

Pel que fa a la segona part, els recursos se centren en desenvolupar nous sistemes, sense coneixement previ d'aquests, amb la qual cosa les estimacions de temps són menys acurades. En finalitzar la part 1 del desenvolupament, es va tornar a analitzar l'estat del projecte i es va generar una nova planificació. A continuació (Taula 4-2) es pot veure el diagrama de Gantt corresponent a la segona part del desenvolupament.

		Duration	Beggning week	End week
D1.1	Rubric 1	1	24	24
D1.2	Rubric 2	1	32	32
D1.3	Rubric 3	1	38	38
D1.4	TFG Delivery	1	40	40
D1.5	Defense	2	43	44
WP3	Development - Part 2	32	25	40
T1.1	Project	17	24	40
T1.2	Library "dll"	2	24	25
T1.3	Split editor/library	1	26	26
T1.4	Play mode	1	27	27
T1.5	Event system	3	28	30
T1.6	Window item	1	31	31
T1.7	Movement animations	2	32	33
T1.8	Button feedback	1	34	34
T1.9	Text font+size change	1	35	35
T1.10	Polish	3	38	40
M1.1	Minimal base core	1	30	30
M1.2	Animations	1	33	33
M1.3	Finished core	1	35	35
WP2	Documentation - Part 2	23	1	25
T1.1	Summary	1	23	23
T1.2	KeyWords	1	23	23
T1.3	Introduction	1	23	23
T1.4	Context	1	24	24
T1.5	Development	15	23	37
T1.6	Conclusions	1	37	37
T1.7	Review & polish	2	37	38
M1.1	First documentation sketch	1	24	24
M1.2	Finished documentation	1	37	37



[Taula 4-2] Diagrama de Gantt - Segona part

4.1 Anàlisi DAFO

En aquest apartat es presenta un anàlisi DAFO, és a dir, un anàlisi dels factors interns i externs relacionats amb el projecte.

Fortaleses

- Experiència en programació d'interfície d'usuari: coneixement de les necessitats.
- Experiència en gestió de projectes i estimació de tasques

Debilitats

- Escassetat de recursos.
- Primera vegada gestionant un projecte en la seva totalitat.

Oportunitats

- Hi ha un espai en els videojocs per a un programa d'aquest tipus.
- No es coneix de cap equip que estigui desenvolupant un programa semblant al d'aquest projecte.

Amenaces

- Un equip més preparat pot fàcilment sobrepassar les capacitats d'aquest projecte.
- Cada vegada es desenvolupen més jocs en Unity i Unreal Engine, que ja compten amb editors de UI

4.2 Riscos i pla de contingències

Tenint en compte la mida del projecte i els recursos disponibles, el principal i únic risc d'aquest era no poder acabar el desenvolupament del programa a temps. Donat que hi havia tasques que no s'havien realitzat mai i era difícil estimar la seva durada, en el cas de necessitar dedicar-li una major quantitat de temps a aquest tipus de tasques, hi havia el risc de no poder acabar el projecte. Per tal d'evitar-ho, es van estructurat les tasques tenint com a objectiu assolir un prototip que sigui totalment funcional. Es van llistat les tasques necessàries per a aconseguir-ho i, a l'hora de planificar, es va donar més marge de temps a les que tenien un major risc.

4.3 Anàlisi inicial dels costos

Els únics costos de desenvolupament d'aquest projecte són de recursos humans. S'ha fet un llistat de totes les tasques a realitzar, amb una estimació de les hores necessàries. També s'ha afegit un percentatge de risc, segons la dificultat de la tasca i si ha experiència prèvia en aquell tema, per anticipar una possible desviació de les hores estimades. A la taula de la següent pàgina (Taula 4-3) es pot veure el llistat de totes les tasques amb el seu cost, a 15€ l'hora (preu l'hora aproximat d'un programador) .

Tasca	Hores estimades	Risc	Desviació	Cost (15€ / h)
Documentació				
Estat de l'art	35	25%	43,75	656,25 €
Planificació	10	0%	10	150,00 €
Metodologia	2	0%	2	30,00 €
Introducció	1	0%	1	15,00 €
Context	3	25%	3,75	56,25 €
Desenvolupament	30	50%	45	675,00 €
Desenvolupament				
OpenGL + SDL base	20	25%	25	375,00 €
Teclat + mouse input	6	25%	7,5	112,50 €
Widgets basics 1	25	50%	37,5	562,50 €
Sistema de colors	3	0%	3	45,00 €
Sistema de conts	10	25%	12,5	187,50 €
Widgets basics 2	5	50%	7,5	112,50 €
Editor basic	25	50%	37,5	562,50 €
Carregar/Guardar escenes	15	25%	18,75	281,25 €
Editor: docking	15	75%	26,25	393,75 €
Editor: jerarquia	5	0%	5	75,00 €
Editor: gizmos	10	25%	12,5	187,50 €
Transformacions 2D	20	75%	35	525,00 €
Editor: selecció	3	0%	3	45,00 €
Jerarquia: control de items	20	0%	20	300,00 €
Implementar llibreria .dll	20	100%	40	600,00 €
Separar editor / llibreria	8	25%	10	150,00 €
Mode de simulació	4	25%	5	75,00 €
Sistema d'events	35	75%	61,25	918,75 €
Widget panell	10	25%	12,5	187,50 €
Animacions de moviment	25	50%	37,5	562,50 €
Feedback del botó	15	25%	18,75	281,25 €
Edició de text avançada	12	25%	15	225,00 €
Total	**392**		**556,5**	**8.347,50 €**

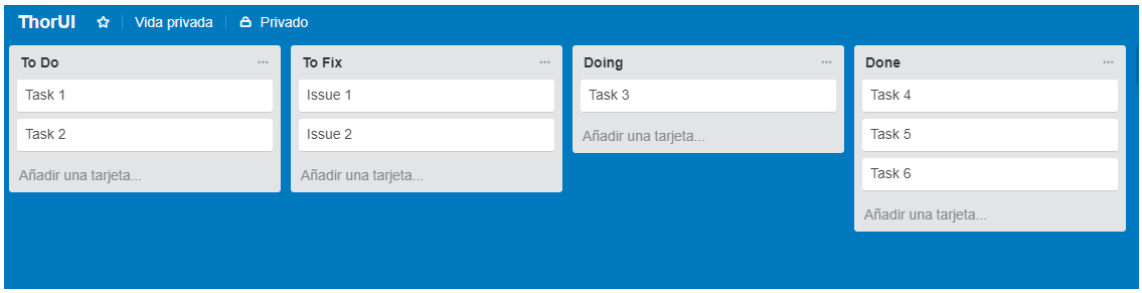
[Taula 4-3] Costos del projecte

4.4 Eines per a la gestió

Per organitzar totes les tasques del projecte s'ha utilitzat, com s'ha vist anteriorment, un diagrama de Gantt mitjançant els fulls de càlcul de Google Drive. Aquest diagrama plasma un esquema general de l'ordre i la durada de realització de cada tasca, des de l'inici fins al final del projecte, per tal d'assegurar tenir totes les funcionalitats a temps.

Paral·lelament s'ha usat l'eina Trello, on es desglossava cada tasca en petits passos a resoldre, de manera que es pogués fer un seguiment més acurat de la feina. A continuació (Fig. 4-1) es pot veure un exemple de la interfície de Trello.

Es pot accedir a aquest programa directament des de la seva pàgina oficial (B11).



[Fig. 4-1] Interfície de Trello

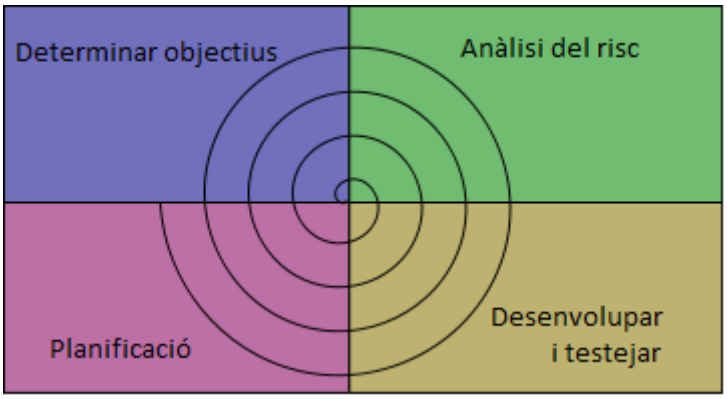
5. Metodologia

La metodologia utilitzada durant el desenvolupament del projecte és l'anomenada desenvolupament en espiral. Es tracta d'un model en bucle, en què cada iteració està formada per un grup d'activitats relacionades entre elles. Les fases de cada iteració són les següents: definir els objectius, analitzar el risc, desenvolupar i provar el bon funcionament i, finalment, planificar una nova iteració.

A cada finalització d'una iteració es comprova que es tenen les noves funcionalitats implementades al projecte, sense haver perjudicat a la resta del programa. En programació, és comú implementar una nova funcionalitat i que aparegui un error en alguna altra part del programa. Una iteració no es considera acabada fins que tots els errors que hagi pogut generar aquesta siguin arreglats.

Donat que ja es tenien llistades les diferents tasques necessàries per acabar el projecte, cada cicle consta en agafar la següent tasca a desenvolupar i desglossar-la en petits passos a seguir, per tal de poder verificar que es van assolint, fins a acabar la nova tasca.

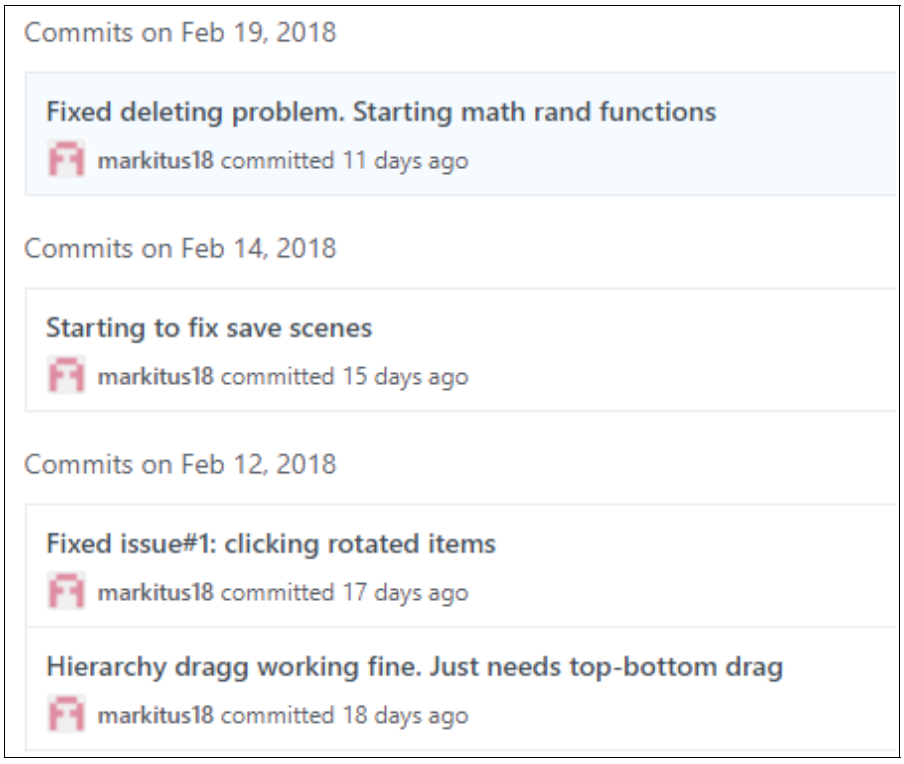
A continuació (Fig. 5-1) es pot veure un esquema que representa un cicle del desenvolupament en espiral.



[Fig. 5-1] Desenvolupament en espiral

5.1 Eines per al seguiment del projecte

L'eina principal de seguiment del projecte és Github, una plataforma pensada per allotjar projectes i que equips grans puguin treballar simultàniament amb els mateixos fitxers. S'utilitza principalment per allotjar codi per a programes. És una eina que s'ha utilitzat constantment al llarg del grau i de la qual ja se'n té força coneixement, així que no ha sigut necessari un aprenentatge previ de les seves funcionalitats abans de començar a desenvolupar el projecte. En el següent enllaç es pot trobar el projecte allotjat a Github: Github – ThorUI. A continuació (Fig. 5-1) es pot veure un exemple de la interfície de Github, on es llisten el que s'anomenen *commits*, una actualització del codi del projecte allotjat als seus servidors.

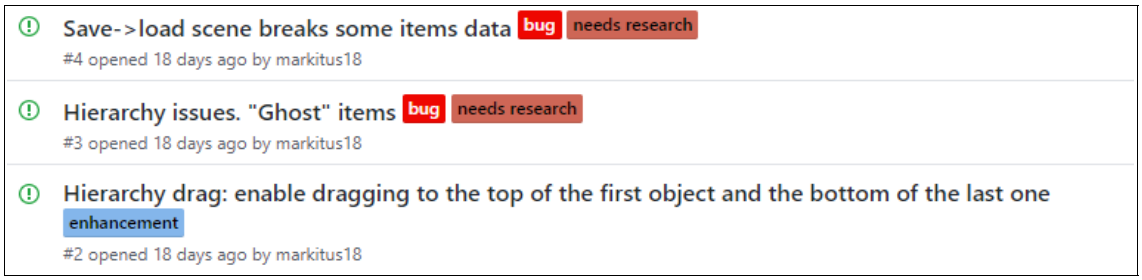


[Fig. 5-2] Exemple de commits de Github

També s'ha utilitzat de forma regular l'eina que proporciona Github, els *issues*, que serveixen per llistar els diferents problemes que s'hagin pogut detectar en el programa. D'aquesta manera, quan es detecta un error, es pot afegir a aquesta llista per solucionar-lo quan sigui més adient, sense necessitat d'interrompre una tasca que s'estigui duent a terme.

Donat que la planificació de les tasques s'ha realitzat amb un marge extra de temps per a realitzar cada tasca, en el cas d'acabar-la abans del previst s'utilitzava aquest temps extra per a solucionar errors que s'haguessin pogut detectar.

Exemple del llistat d' *issues* de Github (Fig. 5-3)



[Fig. 5-3] Issues de Github

5.2 Eines de validació

Per tal de validar que s'assoleixen les diferents tasques, en considerar que ja s'ha implementat la nova funcionalitat, es realitzen un conjunt de tests sobre aquesta mateixa, per tal d'assegurar que funciona correctament sobre diferents situacions. També és necessari realitzar proves amb les funcionalitats que ja existien abans de començar l'última tasca, per comprovar que no se n'hagi vist afectat cap aspecte.

Per altra banda, per assegurar que el disseny del programa és senzill i intuïtiu, durant el desenvolupament del projecte s'han realitzat algunes sessions amb persones externes, on se'ls demanava realitzar una tasca determinada amb un seguit d'instruccions.

Finalment, donat que s'han implementat sistemes ja existents en altres programes, principalment Unity i Unreal Engine, per a comprovar el correcte funcionament d'aquests sistemes es comparava el resultat amb la funcionalitat existent en un dels dos programes de referència, per acabar d'ajustar els diferents matisos, principalment de disseny.

6. Desenvolupament I – La llibreria

En aquest apartat s'explica com s'ha dut a terme el desenvolupament de la llibreria, és a dir, la part del codi que gestiona els elements que apareixen per pantalla (els widgets) i la interacció de l'usuari amb aquests. Tant aquest apartat com el següent, el desenvolupament de l'editor, són altament tècnics, ja que es tracta de la implementació a nivell de codi de les diferents característiques del programa. Tot i així, s'explica de manera que aquells que no tenen coneixements de programació puguin fer un seguiment de les diferents tasques i entenguin com s'han dut a terme.

6.1 Nucli de la llibreria I - OpenGL

Donat que els recursos disponibles per a aquest projecte són escassos i el temps és molt limitat, s'ha decidit utilitzar llibreries ja conegudes prèviament en altres projectes. La primera d'elles és OpenGL (Open Graphics Library).

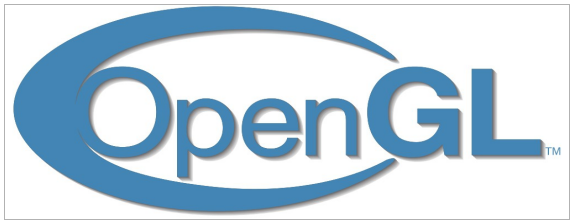
La definició de la organització que l'està desenvolupant actualment és la següent: "OpenGL is the name for the specification that describes the behavior of a rasterization-based rendering system. It defines the API through which a client application can control this system. The OpenGL rendering system is carefully specified to make hardware implementations allowable." (B12).

En altres paraules, OpenGL és una API (Application Program Interface), es a dir, un conjunt de codi semblant a una llibreria, que proporciona utilitats per a generar imatges 2D i 3D. És, de fet, la llibreria de gràfics més àmpliament utilitzada a tot el món.

Donat que la major part de la implementació d'OpenGL tindrà lloc en el codi del joc que utilitzi la llibreria d'aquest projecte, en la pròpia llibreria es fa servir en poques ocasions, per carregar imatges i mostrar els widgets a la pantalla. Així doncs, en aquest projecte no s'entra en detall a explicar la implementació d'OpenGL.

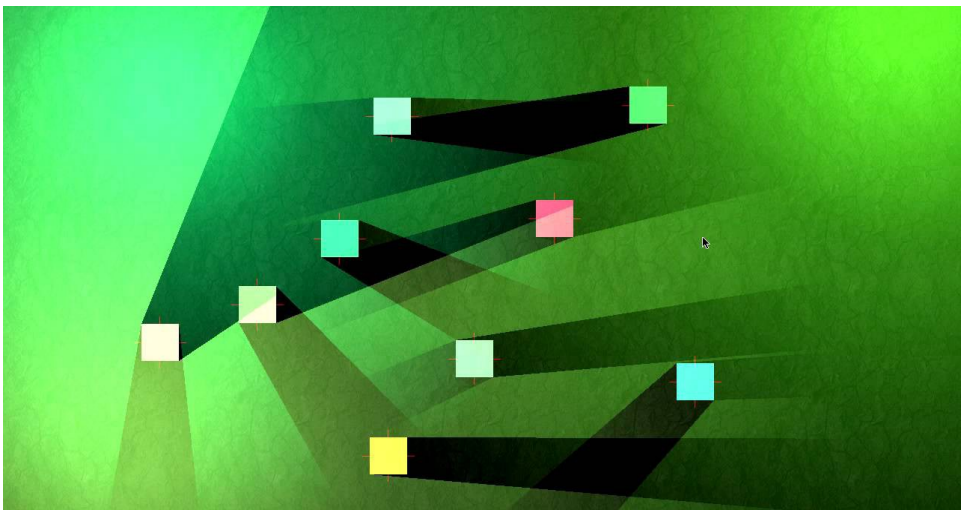
Per a qualsevol interessat en aprendre OpenGL, la web B13 consta amb un seguit de tutorials molt detallats per aprendre'n les bases. També es pot trobar informació addicional sobre OpenGL a la seva web oficial (B14).

A continuació (Fig. 6-1) es pot veure el logotip d'OpenGL i, a la següent pàgina (Figures 6-2 i 6-3), exemples d'imatges aconseguides amb OpenGL.



[Fig. 6-1] Logotip d'OpenGL

[Fig. 6-2] Exemple d'OpenGL en 3D



[Fig. 6-3] Exemple d'OpenGL en 2D

6.2 Nucli de la llibreria II – SDL

SDL (Simple DirectMedia Layer) es l'altra llibreria situada al nucli d'aquest projecte. Aquesta és la definició que es pot trobar a la seva pàgina oficial:

"Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D." (B15*)*

SDL és, doncs, una llibreria que proporciona un conjunt d'utilitats en quant a input (teclat, ratolí i joystick), gestió d'àudio i de la finestra del programa i també eines per a mostrar imatges per la pantalla.

En el desenvolupament de la llibreria de UI SDL s'ha utilitzat per a gestionar l'input del teclat i el ratolí, per carregar imatges i fonts i per carregar i guardar fitxers d'escena. Tots aquests conceptes s'expliquen més detalladament en altres apartats.

Igual que amb OpenGL, en aquest projecte no s'explica extensament l'implementació d'SDL, donat que no es centra en l'ús d'aquestes dues llibreries.

Per a més detalls sobre com implementar SDL, la web B16 consta amb un seguit de tutorials on, pas per pas, s'explica tot el necessari per entendre els usos d'aquesta llibreria.

A continuació (Fig. 6-4) es pot veure el logotip d'SDL i, a la següent pàgina (Figures 6-5 i 6-6), dos exemples de videojocs que utilitzen SDL.

[Fig. 6-4] Logotip d'SDL

[Fig. 6-5] Videojoc Memoria a partir d'SDL

[Fig. 6-6] Videojoc My Tribe a partir d'SDL

6.3 Sistema d'input

En aquest apartat s'explica com s'ha generat el sistema per detectar l'entrada de dades de l'usuari a través del teclat i del ratolí. Com ja s'ha comentat en l'apartat anterior, SDL proporciona eines per a rebre aquestes dades introduïdes per l'usuari. Per a facilitar l'accés a aquestes des dels diferents components de la llibreria de UI, s'ha construït un sistema per sobre d'SDL, que recull les dades i, a través de funcions, permet llegir-les. Al llarg de l'apartat es mencionen diferents funcions d'SDL utilitzades en el projecte i es proporciona el link a la documentació sobre aquestes funcions.

Tant pel teclat com pel ratolí, s'ha creat una enumeració per guardar l'estat de cada botó: tres pel ratolí i un per a cada tecla del teclat. Mes endavant (Fig. 6-7) es pot veure aquesta enumeració. A través de la funció SDL_GetKeyboardState (B17), SDL retorna un vector ple de zeros i uns, segons si la tecla està premuda o no. A partir d'aquest vector, s'omple un altre vector que conté un "Key_State" per a cada tecla. Els valors d'aquest vector dependran de si la tecla està premuda en aquell frame i si ho estava o no en el frame anterior. En la Figura 6-8 es pot veure la funció que se'n carrega d'omplir aquest nou vector a partir de les dades d'SDL. Aquesta s'haurà d'executar a cada frame per tenir actualitzades les dades de l'usuari en tot moment.

```
enum Key_State
{
    KEY_IDLE,
    KEY_DOWN,
    KEY_REPEAT,
    KEY_UP
};
```

[Fig. 6-7] Enum. tecles

```
void UpdateKeyboardState()
{
    const Uint8* keys = SDL_GetKeyboardState(nullptr);
    for (unsigned int i = 0; i < SDL_NUM_SCANCODES; ++i)
    {
        if (keys[i] == 1)
        {
            keyboard[i] = (keyboard[i] == KEY_DOWN || keyboard[i] == KEY_REPEAT) ? KEY_REPEAT : KEY_DOWN;
        }
        else if (keyboard[i] != KEY_IDLE)
        {
            keyboard[i] = (keyboard[i] == KEY_REPEAT || keyboard[i] == KEY_DOWN) ? KEY_UP : KEY_IDLE;
        }
    }
}
```

[Fig. 6-8] Funció per actualitzar el vector de tecles

Pel que fa al ratolí, també es genera un vector per guardar les dades dels botons. Tot i així, la manera d'obtenir-ne les dades a partir d'SDL és diferent que amb el teclat. Per a fer-ho s'utilitza la funció SDL_PollEvent (B18). Aquesta funció omple dades de l'ultim esdeveniment rebut i retorna 0 en el moment en que no n'hi ha cap de nou, així que s'utilitza en bucle, obtenint dades de tots els esdeveniments, fins que retorna 0. A partir d'aquestes dades, es pot identificar si el tipus d'esdeveniment és de ratolí. De la mateixa manera que amb el teclat, s'omple un vector per recollir la informació.

A continuació (Fig. 6-9) es pot veure la funció que guarda les dades dels botons del ratolí.

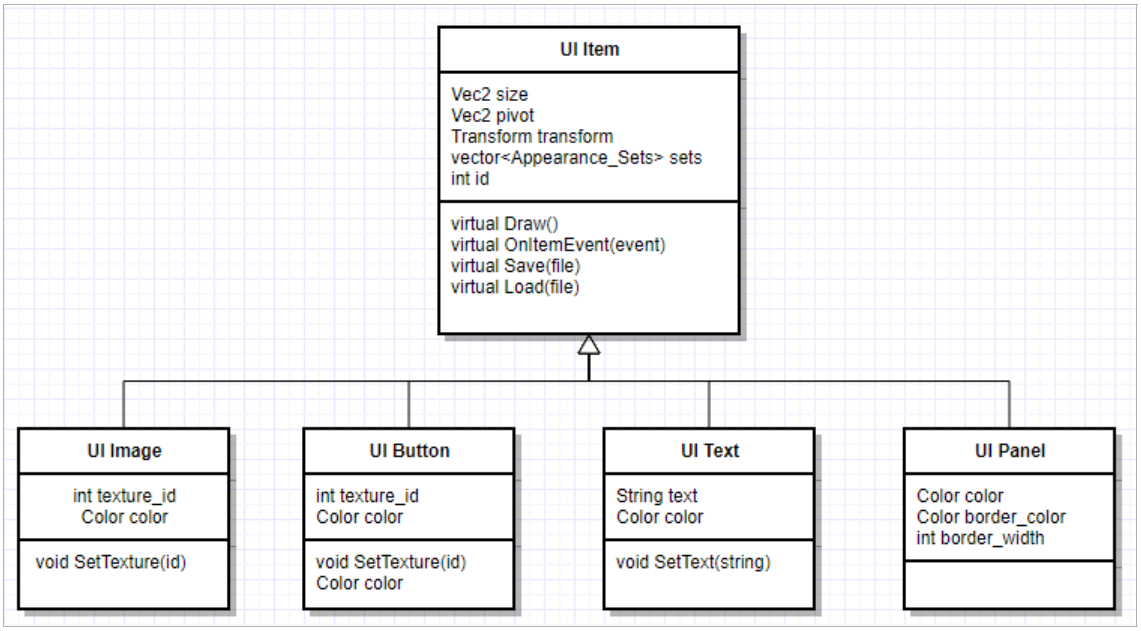
```
while (SDL_PollEvent(event) != 0)
{
    switch (event->type)
    {
        case(SDL_MOUSEBUTTONUP):
        {
            mouse_buttons[event->button.button - 1] = KEY_UP;
            mouse_button_event[event->button.button - 1] = true;
            break;
        }
        case(SDL_MOUSEBUTTONDOWN):
        {
            mouse_buttons[event->button.button - 1] = KEY_DOWN;
            mouse_button_event[event->button.button - 1] = true;
            break;
        }
    }
}
```

[Fig. 6-9] Funció per a guardar els botons del ratolí

Finalment, per guardar la posició del ratolí respecte a la finestra de l'aplicació, simplement cal cridar a la funció SDL_GetMouseState(int& x, int& y) (B19) que omple les dues variables amb la posició del ratolí.

6.4 Estructura dels Widgets

En aquest apartat s'explica quina es l'estructura de codi que s'ha seguit per als widgets que es visualitzen per pantalla i quina és la funció de cadascun. Com s'ha parlat en un apartat previ, aquesta llibreria és de mode retingut, es a dir, cada widget és un objecte que pertany a la classe que li correspongui. En aquest cas s'ha seguit una estructura a partir d'una classe base, *UI Item*, de la qual n'hereten els diferents i widgets. En el següent UML (Fig. 6-10) se'n pot veure l'estructura.



[Fig. 6-10] UML dels Widgets

Així doncs, la classe *UI Item* compta amb un seguit de variables i mètodes comuns en tots els widgets. Algunes d'aquestes funcions són virtuals, de manera que cada widget derivat en fa la seva implementació concreta. Els diferents grups de funcions s'expliquen en els següents apartats. A continuació es llisten els diferents tipus de widgets que s'han creat en aquesta llibreria i es fa una breu explicació de les seves característiques.

UI Image

Aquest widget té una funcionalitat molt simple. Té un color de fons i se li pot aplicar una textura, una imatge 2D. En el cas de no tenir cap imatge, es visualitza com un quadrat del color que tingui assignat. Si, en canvi, té imatge i color, es visualitza la imatge amb el color com a tint. En la següent figura (Fig. 6-11) hi ha un exemple de UI Images vistes des de l'editor.

[Fig. 6-11] Exemple de UI Images

UI Button

El botó consta amb la mateixa funcionalitat que la imatge, amb l'afegit que l'usuari el pot prémer. Els botons, per defecte, tenen un UI Text dins que, si es vol, es pot treure. En la Figura 6-12 es poden veure dos botons, un amb textura i l'altre sense.

[Fig. 6-12] Exemple de UI Buttons

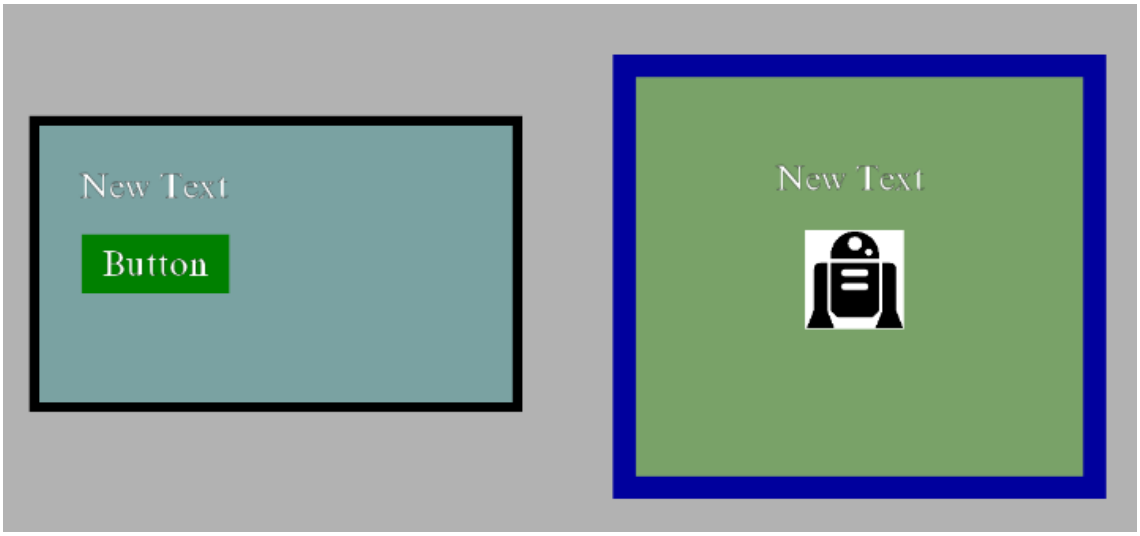
UI Text

El text consta d'un conjunt de caràcters, que es visualitzaran per pantalla segons el color que se li assigni. És possible canviar-ne la font i la mida de la tipografia. A continuació (Fig. 6-13), dos exemples de textos en l'editor.

[Fig. 6-13] Exemple de UI Texts

UI Panel

Aquest widget es tracta d'un rectangle d'un color i un marge al voltant, d'un altre color. Es pot definir la mida d'aquest marge així com els dos colors. Està pensat per a agrupar altres widgets dins seu. En la Figura 6-14 s'hi troba un exemple de UI Panels, amb diferents widgets a dins.



[Fig. 6-14] Exemple de UI Panels

6.5 Matrius de transformacions

En aquest apartat es comenta com s'ha dut a terme la implementació de les matrius de transformació en els widgets. Primer de tot, cal explicar què és una matriu de transformació. Donat que són conceptes estrictament matemàtics, no s'explicarà què és una matriu ni com es realitzen les diferents operacions entre elles.

Una matriu de transformació és una matriu (un grup de números organitzats en files i columnes) que s'utilitza per a representar un eix de coordenades en un lloc de l'espai. Per a simplificar-ho, i en aquest context, es pot entendre una matriu de transformació com una representació d'una posició, una rotació i una escala en l'espai. A través de la multiplicació de matrius es pot aconseguir desplaçar, rotar o escalar una matriu base, que representa un widget a la pantalla. Donat que la UI d'aquest projecte es en 2D, les matrius necessàries per a representar-les també seran de dues dimensions. En la següent Figura (Fig. 6-15) s'hi troba un exemple de matriu de translació, un de matriu d'escala i un de matriu de rotació.

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[Fig. 6-15] Matrius de translació, escala i rotació

Inicialment en el projecte les dades dels widgets es guardaven en posició, escala i rotació separadament, sense fer ús de matrius. Es va veure que aquesta manera d'estructurar les dades no era la millor opció i portava alguns problemes, així que es va decidir passar a utilitzar matrius de transformació, que també evitaven possibles errors en un futur.

Una vegada decidit que s'utilitzarien les matrius de transformació, es van presentar varies maneres d'implementar aquest sistema:

La primera va ser la de fer servir la llibreria MathGeoLib, ja que hi havia coneixement previ sobre aquesta i això estalviava temps a l'hora d'implementar-la. Tot i això, hi havia dos aspectes que van fer que es descartés el seu ús. El primer va ser que aquesta llibreria utilitza matrius per a fer representacions en tres dimensions, així que s'hauria d'adaptar el codi d'alguna manera per a passar-ho a dues dimensions, cosa que implicaria un codi pitjor estructurat i menys simplificat. El segon aspecte va ser que MathGeoLib consta amb una gran quantitat de funcionalitats que, per a representar widgets en una pantalla en dues dimensions, no són necessàries. Totes aquestes funcionalitats extres, que no es farien servir, comportaven un afegit en quant a quantitat de codi i, per tant, la mida de la llibreria i de l'editor seria més gran. Finalment, doncs, es va descartar aquesta opció.

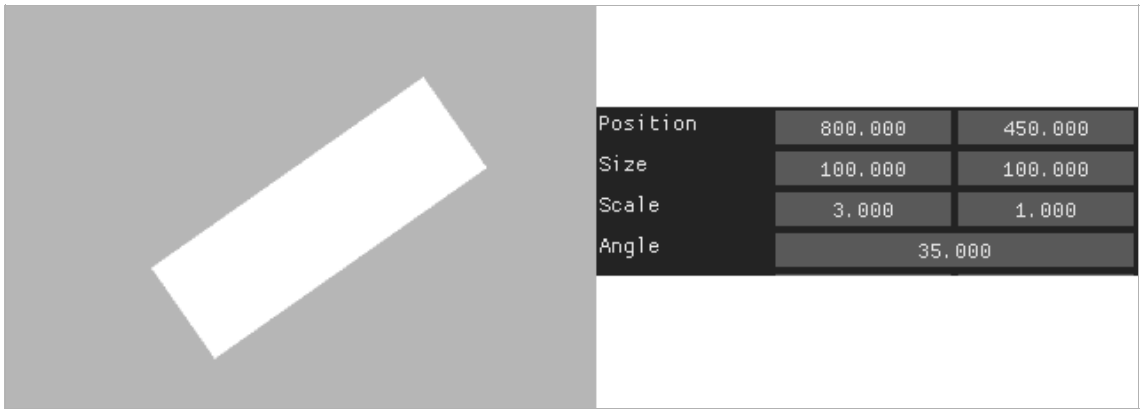
La segona possibilitat va ser la de buscar una altra llibreria que proporcionés funcionalitats per a matrius en dues dimensions. La primera que es va trobar va ser la de Box2D. Tot i ser un motor de joc, té algun fitxer de matrius en dues dimensions que podia ser útil. Revisant el codi es va veure que hi faltaven algunes funcionalitats i que, per tant, s'hauria d'afegir més codi per sobre del seu. En veure això es va pensar que, molt possiblement, no hi hauria cap llibreria ja existent que s'ajustés completament a les necessitats d'aquest projecte.

Finalment, doncs, es va arribar a una nova opció, més atrevida que les anteriors. Implementar tot el sistema de matrius de transformació des de zero, començant pels càlculs (multiplicacions, sobretot) de matrius. Aquesta nova possibilitat arribava amb tot un seguit d'avantatges: el sistema s'ajustaria totalment a les necessitats del projecte, ja que estaria creat específicament per a aquest. Si, en un futur, es decidia ampliar-lo, es podria fer de manera més estructurada i més ràpida, ja que es coneixeria la totalitat del codi. L'últim avantatge, potser el més gran de tots, va ser el d'ampliar el coneixement sobre matrius. Les matrius es fan servir en la gran majoria de videojocs, així que saber com crear un sistema així i entendre'l és una gran eina per a un programador. Així doncs, es va decidir per aquesta opció, i implementar el sistema des del principi.

En aquest treball no s'entrarà a explicar en detall les bases dels càlculs de matrius, ja que és implementació d'aritmètica en codi. El que sí que s'explicarà, però, és la implementació d'aquest sistema concretament per als widgets.

Per a aplicar matrius de transformació als widgets s'ha creat la classe *Transform*. Aquesta classe té diferents capes de complexitat, que s'explicaran una per una.

Començant per una simple transformació, primer es guarden les dades de posició, rotació i escala en variables separades. Per generar aquesta matriu de transformació només cal aplicar les matrius vistes anteriorment en la Figura 6-15 a una matriu identitat, amb els valors corresponents a les variables de posició, rotació i escala. Amb això s'aconsegueix tenir una primera representació del widget en la pantalla a través d'una matriu. En la Figura 6-16 es pot veure el resultat, amb la seva matriu corresponent (Fig. 6-17).

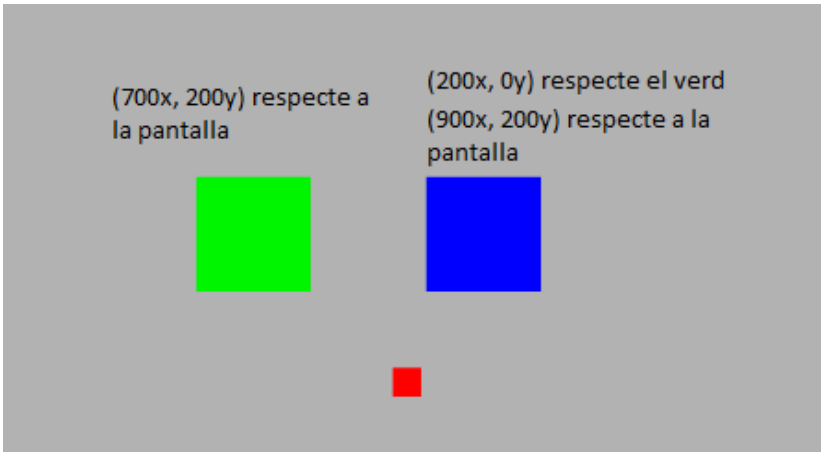


[Fig. 6-16] Imatge amb posició, rotació i escala

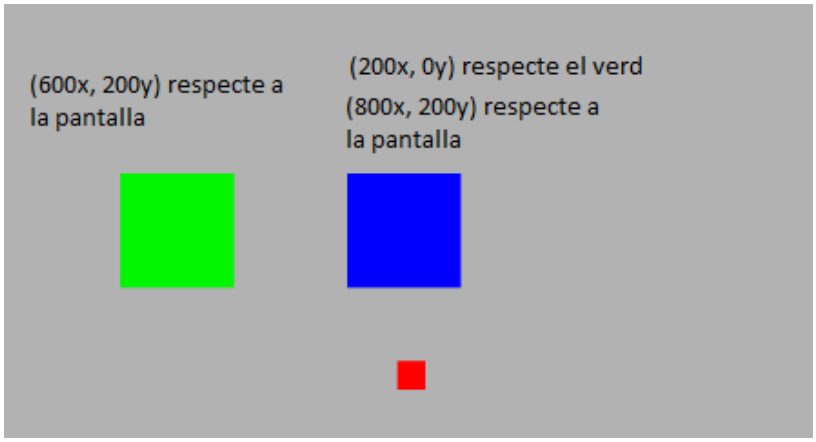
2.46	-0.57	800.00
1.72	0.82	450.00
0.00	0.00	1.00

[Fig. 6-17] Matriu corresponent a la imatge de la Fig. 6-16

El següent pas és introduir la jerarquia de widgets. Per aquells que no estiguin familiaritzats amb l'ús de motors de joc, una jerarquia es pot entendre com una relació de pare-fill, on el fill té una matriu de transformació en respecte al seu pare. La manera més fàcil d'entendre aquest concepte és amb un exemple. El widget blau de la Fig. 6-18 és el fill del widget verd. La posició del blau, doncs, és en respecte al verd. Com es pot veure en la Fig. 6-18, el widget verd és en la posició (700x, 200y). El blau té posició de (200x, 0y), així que es troba 200 píxels a la dreta del verd, donat que aquest és el seu pare. Per tant, la posició respecte a la pantalla del quadrat blau és (900x, 500y). En el cas que el verd es mogui 100 píxels cap a l'esquerra, el blau també es mourà, mantenint la seva posició respecte al verd. En les Figures 6-18 i 6-19 es pot veure aquest exemple gràficament, amb el quadrat vermell com a referència.



[Fig. 6-18] Jerarquia - Exemple 1



[Fig. 6-19] Jerarquia - Exemple 2

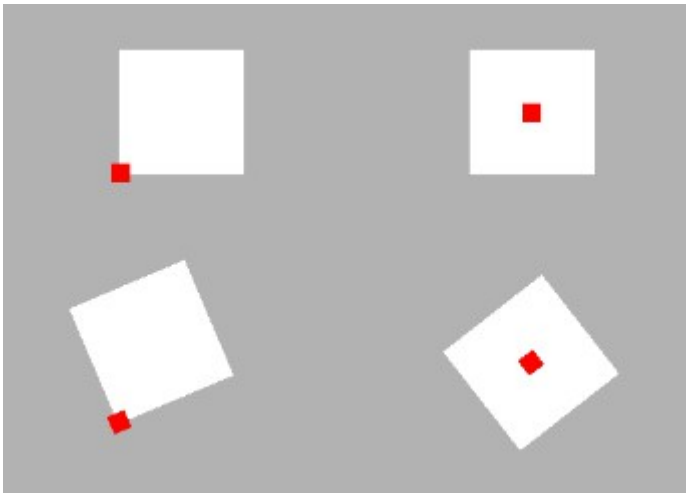
Una vegada explicat el concepte de jerarquia, aplicar-lo a amb matrius de transformació és força senzill. Si bé en l'exemple anterior, per aconseguir la posició en pantalla del quadrat blau, es necessitava una suma (Posició verd + Posició blau = Posició final blau), per passar-ho a matrius és només cal multiplicar la matriu del widget "pare" per la del fill, per aconseguir la relativa a la pantalla. Així doncs, si abans es necessitava una matriu per cada widget, ara se n'utilitzen dues. Una és la matriu local, la que representa la posició respecte al pare, i l'altre la global, la que representa la posició en pantalla.

Per tant, per aconseguir la matriu global de cada widget, s'utilitza la següent formula:

matriu global = matriu global pare * matriu local

A través d'aquest nou sistema de dues matrius s'aconsegueix el resultat vist en la Figura 6-19, en la pàgina anterior.

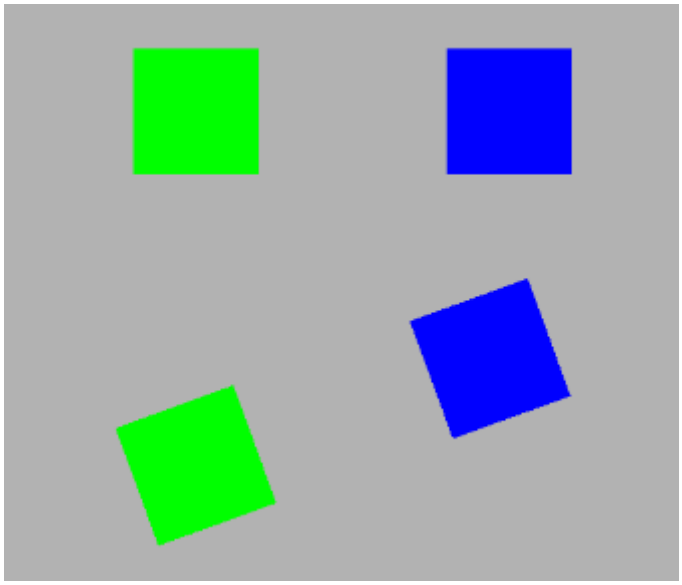
La darrera capa de complexitat és tracta d'introduir el concepte de pivot. Fins ara, la posició dels widgets es considerava al centre del quadrat. Per tant, en rotar-los, sempre es rotaven des d'un eix central. En la Figura 6-20 es pot veure un exemple d'aplicar una rotació utilitzant un pivot o un altre. El quadrat vermell representa la posició del pivot.



[Fig. 6-20] Rotacions amb pivot

El valor del pivot va de 0 a 1, 0 sent la cantonada inferior esquerra i 1 la superior dreta, amb un valor per a cada eix (x i y).

Per implementar el sistema del pivot només cal adonar-se que el resultat és el mateix que s'obté a través del sistema de jerarquia (pare-fill). En el cas de rotar el widget pare, el fill rota utilitzant la posició del pare com a pivot. En el següent exemple (Fig. 6-21) el quadrat verd és el pare del blau. En rotar el verd, el blau rota també, utilitzant el verd com a pivot.



[Fig. 6-21] Rotació a través de jerarquia

Tenint això en compte, la manera d'afegir el pivot al sistema de transformació va ser la d'utilitzar, altra vegada, dues matrius. La matriu local anterior, doncs, es divideix en dues noves matrius. Una primera matriu que representarà la posició del pivot i una altra que representarà el centre del widget.

Una vegada es tenen les dues matrius, també cal utilitzar dos tipus de pivot: un per al widget, i l'altre per les transformacions. El pivot del widget és el comentat anteriorment, que té un valor relatiu, de 0 a 1. El valor per a les transformacions, en canvi, té un valor en píxels de la pantalla. Per a trobar aquest valor, cal utilitzar la següent formula:

pivot transformació = ((0,5, 0,5) – pivot widget) * mida widget

En el cas de tenir un pivot de widget de (0, 0), es a dir, cantonada inferior esquerra, i una mida de (100, 100), el valor en píxels esdevindria el (-50, -50), partint del centre del widget com a origen de coordenades. Aquest valor final és la relació entre la matriu de pivot del widget i la matriu central.

6.6 Guardat d'escenes

La manera d'utilitzar aquesta eina per a un videojoc és utilitzar un programa (l'editor) per dissenyar on es col·loquen els diferents tipus de widgets, quina forma tindran, quina interacció hi haurà entre ells, etc. Però tot això es fa dins de l'editor gràfic. Per a transmetre tota aquesta informació al videojoc, cal guardar totes les dades en un fitxer, que s'utilitzarà més endavant durant l'execució del videojoc.

Es va decidir utilitzar una llibreria ja coneguda de projectes anteriors, la llibreria Parson (B20). Aquesta llibreria inclou un conjunt de funcions per a escriure i llegir dades en un fitxer de tipus JSON, un format de fitxer que permet guardar dades de manera que siguin fàcils de llegir per l'ull humà. Es pot trobar més informació sobre aquest tipus de fitxer a la seva pàgina oficial (B21).

Com s'ha mencionat anteriorment, la llibreria Parson proporciona funcions per a poder escriure en un fitxer JSON, però el procés per a utilitzar-les és una mica rebuscat. Per tal de poder guardar i carregar dades a fitxers de manera senzilla, es va crear una nova classe per a gestionar aquesta llibreria. D'aquesta manera afegir i llegir dades del fitxer es podia fer mitjançant una sola funció. En la imatge de la següent pàgina (Fig. 6-22) es pot veure un fragment d'una escena de UI guardada en un fitxer JSON.

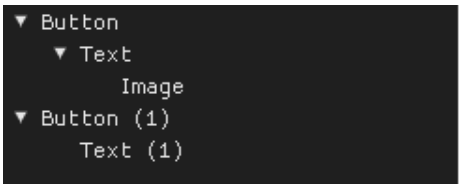
```
"Items": [
    {
        "Name": "OpenButton",
        "Item_Type": 0,
        "Position": [
            680.078918,
            669.329407
        ],
        "Scale": [
            1,
            1
        ],
        "Rotation": 0,
        "Size": [
            161.000015,
            101.000008
        ],
        "Pivot": [
            0.500000,
            0.500000
        ],
        "ID": 3348,
        "Parent ID": 0,
        "Color": [
            0.615686,
            0,
            0,
            1
        ],
```


[Fig. 6-22] Fragment d'un fitxer JSON

En aquest exemple es pot veure les dades d'un widget, de tipus UI Button. Es pot veure el seu nom, les dades de transformació i el seu color, entre d'altres.

Un problema a solucionar en quant a guardar escenes és el de tenir referències entre widgets. Per exemple, per a guardar les dades de la jerarquia entre widgets, cal saber quin és pare de quin altre. Una de les opcions és guardar aquesta dada utilitzant el nom del widget, però no és un mètode segur, ja que porta problemes en el cas que dos widgets tinguin el mateix nom. Aquí és on s'introdueix el concepte d'identificador únic (UID). De la mateixa manera que cada persona té el seu número de carnet d'identitat i no n'hi ha cap de repetit, a tots els widgets se'ls hi assigna un número únic. D'aquesta manera, per guardar dades com la jerarquia, només cal guardar-ne aquest identificador.

En Figura 6-23 es pot veure el resultat del fitxer corresponent a la jerarquia vista en la Figura 6-24. Per tal de simplificar el fitxer i fer-lo més fàcil de llegir per a aquest exemple, només es mostren les dades importants.



[Fig. 6-24] Jerarquia visualitzada des de l'editor

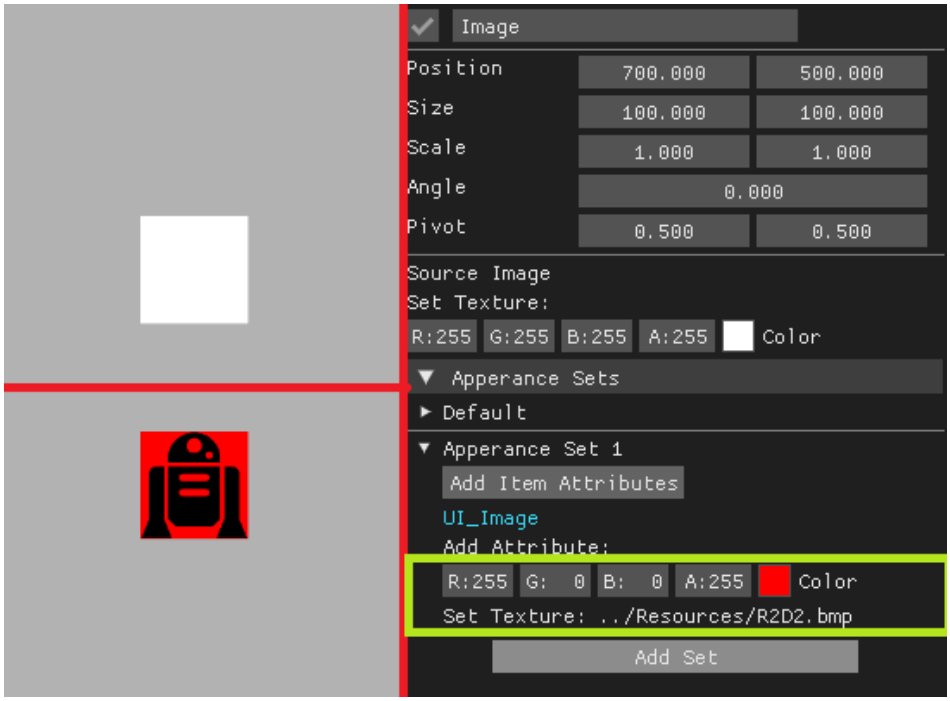

```
"Items": [
    {
        "Name": "Button",
        "Item_Type": 1,
        "ID": 25531,
        "Parent ID": 0,
    },
    {
        "Name": "Text",
        "Item_Type": 2,
        "ID": 796,
        "Parent ID": 25531,
    },
    {
        "Name": "Image",
        "Item_Type": 3,
        "ID": 21651,
        "Parent ID": 796,
    },
    {
        "Name": "Button (1)",
        "Item_Type": 1,
        "ID": 2254,
        "Parent ID": 0,
    {
        "Name": "Text (1)",
        "Item_Type": 2,
        "ID": 15265,
        "Parent ID": 2254,
    },
```


[Fig. 6-23] Fitxer d'escena corresponent a la jerarquia de la figura 6-24

6.7 Appearance Sets (Conjunts d'aparença)

"Appearance Sets" és el nóm que s'ha donat en aquest projecte al conjut de totes les dades d'un widget que formen part del seu aspecte visual. La posició, el color o la rotació formen part d'aquest conjunt, mentres que l'identificador únic no, ja que no afecta a l'estat visual.

En agrupar totes aquestes dades, l'usuari de l'editor pot configurar diferents grups d'aparença per a un mateix widget, de manera que en el fitxer d'escena es guarden les dades actuals del widget i unes dades adicionals que l'usuari hagi introduït. A través d'una crida a una funció, és possible canviar el conjunt d'aparença actual d'un widget per un altre que tingui guardat. En la següent imatge (Fig. 6-25) es pot veure una UI Image de color blanc i sense textura, pero amb un "Appearance Set" guardat, on es veu el rectangle verd. En aplicar aquest conjunt, la imatge canvia automàticament, tot i que segueix sent el mateix widget. D'aquesta manera l'usuari té la possibilitat de mantenir el mateix widget a la pantalla i canviar-ne la aparença al llarg del temps.

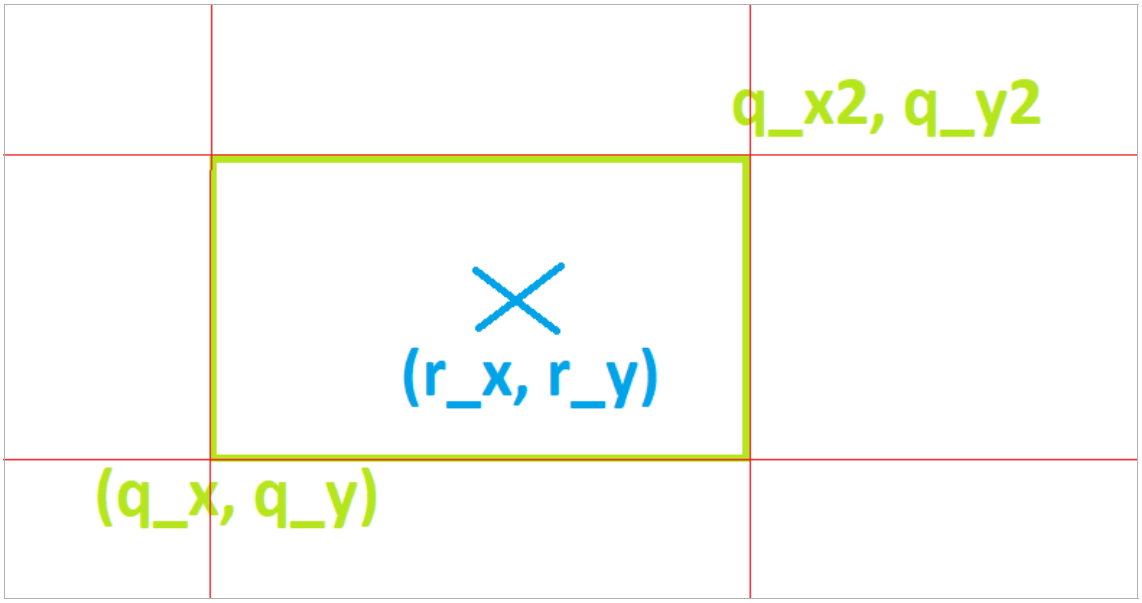


[Fig. 6-25] Conjunt d'aparença

6.8 Sistema d'esdeveniments

En aquest apartat s'explica el sistema que gestiona la interacció de l'usuari del videojoc amb els diferents widgets de la pantalla. Cal implementar com es detecta aquesta interacció i també quines són les seves conseqüències.

Com en la majoria de sistemes en aquest projecte, es van construïnt capes de complexitat una per una, començant per la més bàsica: la intersecció d'un punt amb un quadrat. Donat que tots els widgets són rectangulars, aquesta formula servirà per a detectar en quin moment el ratolí es troba a sobre del widget. Per entendre fàcilment el càlcul, en la Figura 6-26 es pot veure de manera gràfica.



[Fig. 6-26] Intersecció ratolí-quadrat

Per a comprovar que el punt R estigui dins del rectangle Q, r_x ha d'estar entre el rang [q_x, q_x2] i r_y entre [q_y, q_y2]. Si es compleixen les dues condicions significa que el punt esta dins del rectangle. D'aquesta manera ja es pot detectar el ratolí quan entra i quan surt dels widgets. Si el ratolí es troba dins d'un widget i l'usuari prem el botó del ratolí, significa que ha premut el widget.

El següent pas és portar aquesta detecció amb el ratolí a les matrius de transformació. En tenir un widget rotat i escalat no és possible aplicar aquest sistema directament. Per

solucionar aquest problema només cal passar la posició del ratolí a l'eix de coordenades del widget i, després, ja es pot fer la comprovació de la mateixa manera que s'ha vist a l'exemple. Per fer aquest canvi de coordenades només cal multiplicar el punt del ratolí per la matriu inversa del widget. Una vegada es té això, doncs, ja es pot detectar el ratolí sobre qualsevol widget col·locat de qualsevol manera.

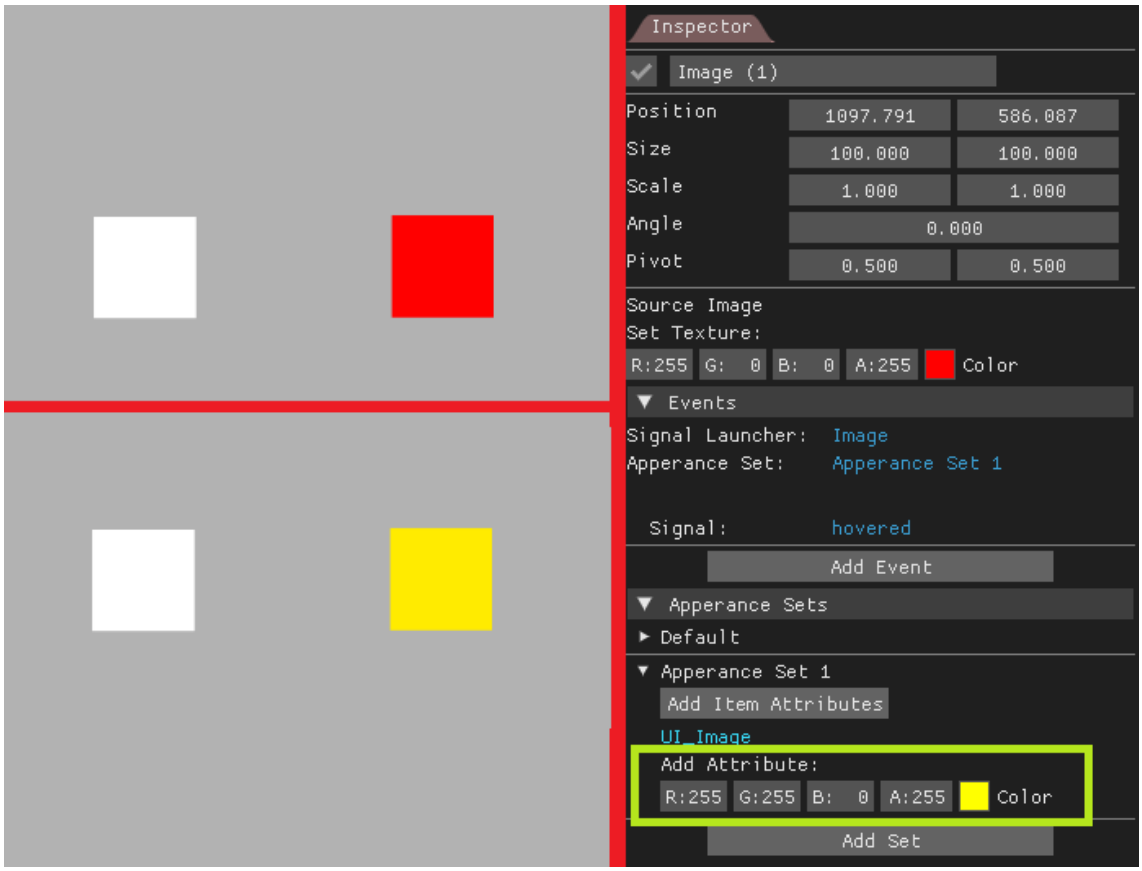
D'aquesta manera la llibreria ja detecta els esdeveniments que l'usuari genera sobre els widgets, però no hi ha cap reacció. Des del codi del videojoc s'han de poder rebre aquests esdeveniments d'alguna manera. Aquí és on entren les lambdes, un concepte de C++. Per a una informació detallada sobre què es una lambda es pot visitar la web B22. Per aquest projecte, i per entendre com s'ha aplicat el seu funcionament, només cal saber que una lambda és una manera de guardar una funció concreta d'un objecte. Com a exemple, tenint un UI Button creat, amb una lambda es pot guardar la funció per canviar-li el color, i cridar-la en un altre moment a través d'aquesta lambda.

A partir d'aquest concepte, s'ha creat una nova classe, la classe *Signal*, que serveix per a guardar múltiples lambdes amb els mateixos tipus d'arguments i cridar a totes les funcions associades en algun moment donat. Cal esmentar, primer, que un widget qualsevol pot generar un esdeveniment en entrar el ratolí a la seva àrea i un altre al sortir-ne. S'ha creat, doncs, una *Signal* associada a cadascún d'aquests esdeveniments de manera que, quan el ratolí entra dins d'un widget, aquesta *Signal* crida a totes les funcions de les lambdes que té guardades.

Tenint el sistema de *Signals* muntat, el programador del videojoc només cal que assigni una funció del seu propi codi a una *Signal* del widget que vulgui i així, en el moment en què el ratolí entri al widget, s'activa la *Signal* i es crida la funció assignada. D'aquesta manera el programador que utilitzi aquesta eina és capaç de rebre tots els esdeveniments dels widgets i gestionar-los com calgui.

Per acabar amb el sistema d'esdeveniments, des de l'editor també es possible establir aquestes connexions de *Signals* entre dos widgets o, fins i tot, un widget amb ell mateix. Es vincula la *Signal* dels widgets amb una funció d'un altre widget que gestiona les senyals que rep. Des de l'editor es pot vincular una senyal rebuda amb un "Appearance

Set", explicat en l'apartat anterior. En tenir aquest vincle, quan un widget rep una senyal d'un altre widget, pot canviar d'aparença automàticament. Amb un exemple la connexió de senyals queda més clara:



[Fig. 6-27] Exemple Senyal - Appearance Set

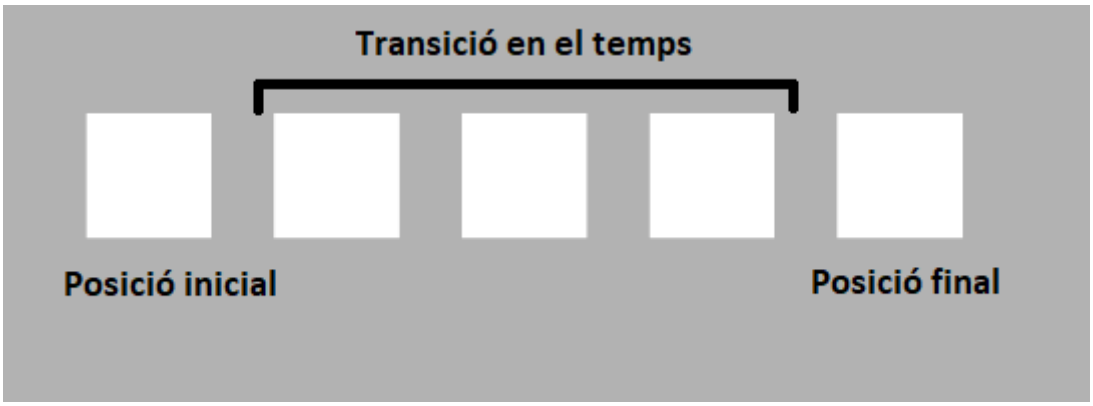
En la Figura 6-27 hi ha inicialment dues UI Images. En la vermella, s'ha creat un event, que rebrà la senyal hovere (es a dir, quan el ratolí entra al widget) de la imatge blanca. Aquesta senyal s'ha vinculat a l'Appearance Set 1, on el color de la imatge és groc. En el moment en què s'executa la simulació i es posa el ratolí sobre la imatge blanca, l'altra es torna groga.

6.8 Animacions

En arribar a aquest punt del projecte, ja és possible generar la interfície d'usuari d'un videojoc de manera completa. Tot i així, però, és molt estàtica. Com s'ha vist en l'apartat anterior, es poden canviar les característiques dels widgets segons la interacció de l'usuari, però això passa de manera instantània i, per tant, no hi ha moviment a la pantalla. En aquest moment és on s'introdueix el concepte de les animacions.

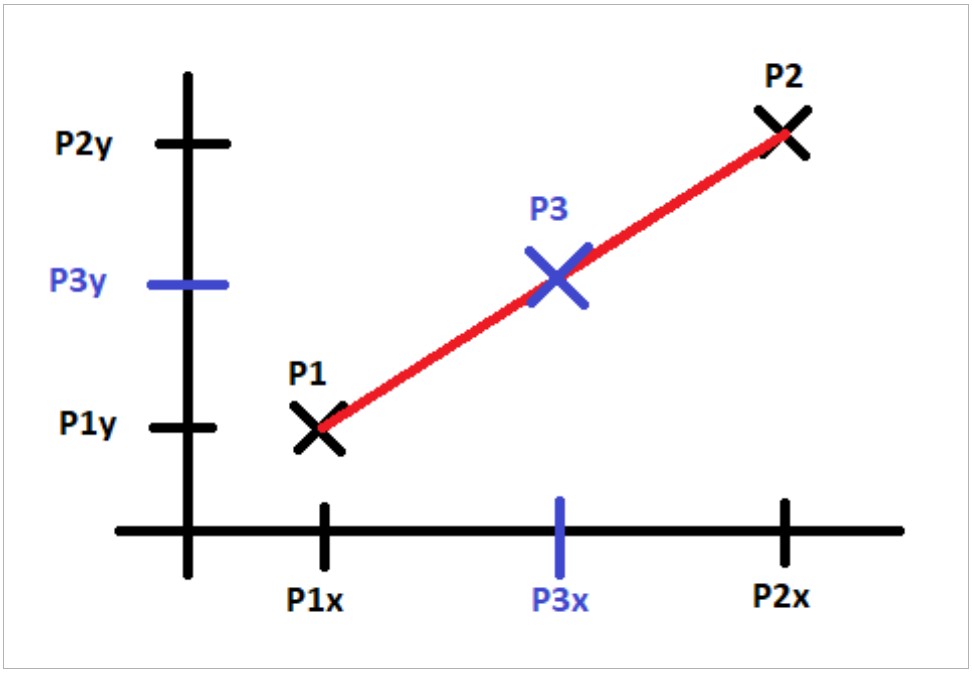
Les animacions són una transició d'un punt a un altre. En comptes de canviar la posició d'un objecte instantàniament, l'objecte es trasllada cap a la nova posició al llarg del temps. Tot i que la posició és la manera més fàcil d'entendre les transicions, també es poden fer amb altres característiques, com el color o la mida. En les dues següents Figures (6-28 i 6-29) es pot veure la diferència en canviar el conjunt d'aparença d'un objecte en què, en aquest cas, només se'n veu afectada la posició.

[Fig. 6-28] Canvi de posició sense animació



[Fig. 6-29] Canvi de posició amb animació

La implementació de les animacions es força sencilla, es tracta d'aplicar el concepte matemàtic d'interpolació. La definició d'interpolació és l'obtenció de nous punts a partir del coneixement prèvi d'un conjunt de punts. En aquest projecte s'utilitza, més concretament, una interpolació lineal. A nivell matemàtic una interpolació lineal s'aconsegueix al traçar una línia entre els dos punts coneguts. Qualsevol altre punt pertanyent a aquella línea s'obté a través de la interpolació. En el següent exemple es pot veure el procés d'una interpolació lineal a nivell gràfic. El punt P3 s'obté a través de la interpolació entre P1 i P2.



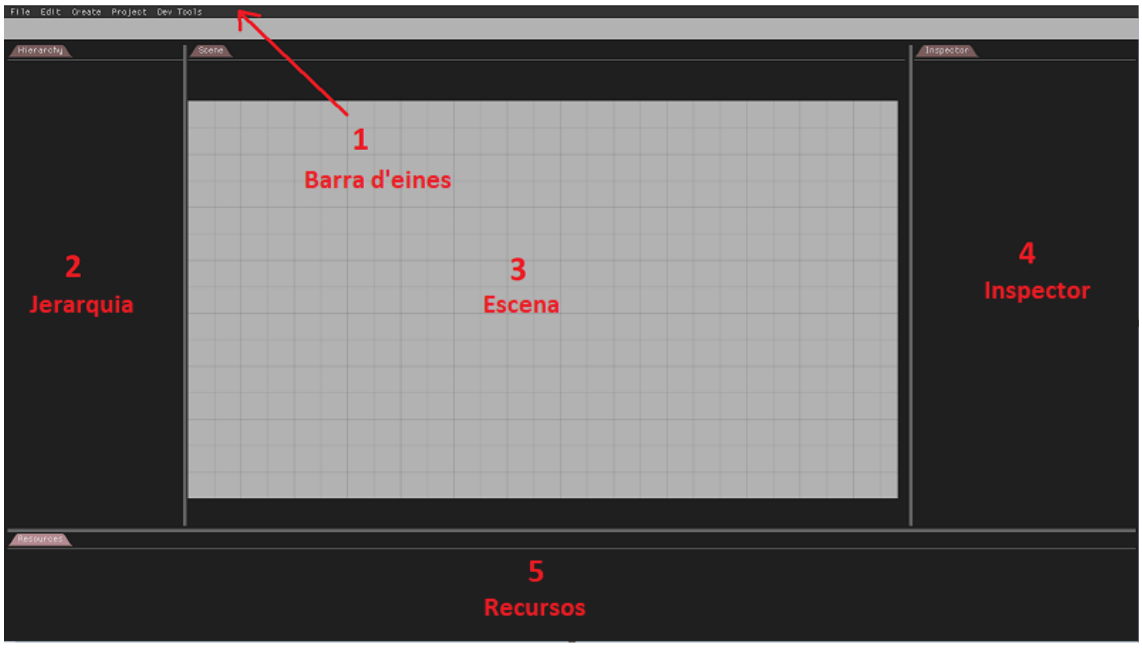
[Fig. 6-30] Interpolació lineal entre dos punts

Per implementar les animacions a través de la interpolació lineal només cal prendre la posició (o qualsevol altra característica del widget) com a valor X i vincular-lo amb la variable Y que, en aquest cas, és el temps. En tenir un widget a la posició (0x, 0y) i indicar que la posició final és la (500x, 0y) al cap de 5 segons, a través de la interpolació lineal s'obté que al segon 2 la posició del widget és (200x, 0y). Aquest és cas la Figura de la pàgina anterior (6-28)

7. Desenvolupament II - L'editor

Com s'ha explicat en la introducció, aquest projecte consta de dues parts. La creació d'una llibreria per a poder programar la interfície d'usuari d'un videojoc i la creació d'un programa per a dissenyar aquesta interfície, utilitzant la llibreria. Tot i que en aquest document s'explica el desenvolupament de l'editor després de les funcionalitats de la llibreria, totes dues coses es van desenvolupar simultàniament, de manera que les noves funcionalitats que s'anaven afegint a la llibreria es poguessin provar des de l'editor.

En aquest apartat s'expliquen les diferents funcionalitats amb què consta l'editor i la seva implementació. En la següent figura (7-1) se'n poden veure les diferents parts.



[Fig. 7-1] Pantalla de l'editor

7.1 ImGui

Tal i com s'ha vist a l'anàlisi de l'estat de l'art, dear-ImGui és una gran llibreria de UI per al desenvolupament d'eines. Donat que la llibreria ja era coneguda de projectes anteriors i satisfeia tots les necessitats d'aquest projecte, es va decidir utilitzar-la per a l'editor. Al llarg de l'apartat del desenvolupament de l'editor s'aniran explicant implementacions específiques d'ImGui, de manera que aquells que ja coneixen la llibreria o que volen començar a utilitzar-la, puguin seguir-ne els passos.

ImGui dóna suport per diferents llibreries de gràfics (DirectX, OpenGL, SDL,...) donat que la llibreria d'aquest projecte ja utilitza SDL, també se'n fa ús amb ImGui ja que així no és necessari fer cap altra implementació.

Començar a utilitzar ImGui és molt senzill. Al principi de l'execució del programa s'ha de cridar la inicialització de la llibreria a través de la funció ImGui_ImplSdlGL3_Init(). Després, en el bucle d'execució, cal cridar a la funció ImGui_ImplSdlGL3_NewFrame() i, al final del frame, ImGui::Render(). Totes les crides a funcions per afegir nous elements de UI a través de ImGui s'han de realitzar entre la funció NewFrame i la funció de Render. En la Figura 7-2 es pot veure, en pseudo-codi, l'estructura que hauria de tenir.

```
int main(int argc, char** args)
{
    SDL_Window* window = nullptr;

    ImGui_ImplSdlGL3_Init(window);

    while (true)
    {
        ImGui_ImplSdlGL3_NewFrame(window);
        /*
            //Crides de creació i gestió de la UI de ImGui
        */
        ImGui::Render();
    }
}
```

[Fig. 7-2] Pseudo-codi estructura ImGui

A partir d'aquesta estructura per començar a generar UI a través d'ImGui només cal fer les crides a funcions corresponents. Per exemple, amb ImGui::Begin() es renderitza una nova finestra, i amb ImGui::Button() apareix un botó dins de la nova finestra. La finestra de test de ImGui és molt útil per aprendre les diferents coses que es poden fer a través d'aquesta llibreria, ja que mostra tot un seguit de característiques i després es pot accedir al codi per a veure com s'implementen. Per tal de mostrar aquesta finestra, només s'ha de cridar la funció ImGui::ShowTestWindow().

7.2 Barra d'eines

La barra d'eines de l'editor, com a la majoria de programes, compta amb un seguit de funcionalitats per ajudar a gestionar el projecte. A continuació s'explica tot el que es pot fer amb aquesta barra i, més endavant, com s'ha implementat.

File (Arxiu)

Aquest menú permet, com el seu nom indica, gestionar fitxers. Té tres opcions: guardar una escena (Save), carregar-ne una altra (Load) o començar-ne una de nova (New Scene). La funcionalitat de gestió d'arxius es troba explicada en l'apartat de Guardat d'escenes (pàg. 52).

Edit (Edició)

- Canvas: permet editar la mida de l'espai designat per a la UI.
- Grid: permet editar l'espaiat de la graella que es pot veure al panell d'escena. En prémer la tecla Ctrl i moure els widgets, la posició s'ajusta a punts de la graella.
- Snap: en prémer la tecla Ctrl mentre s'escala o es rota un widget, els valors van augmentant segons un interval (de 5 en 5, per exemple). Aquest opció permet editar aquests intervals.

Create (Crear)

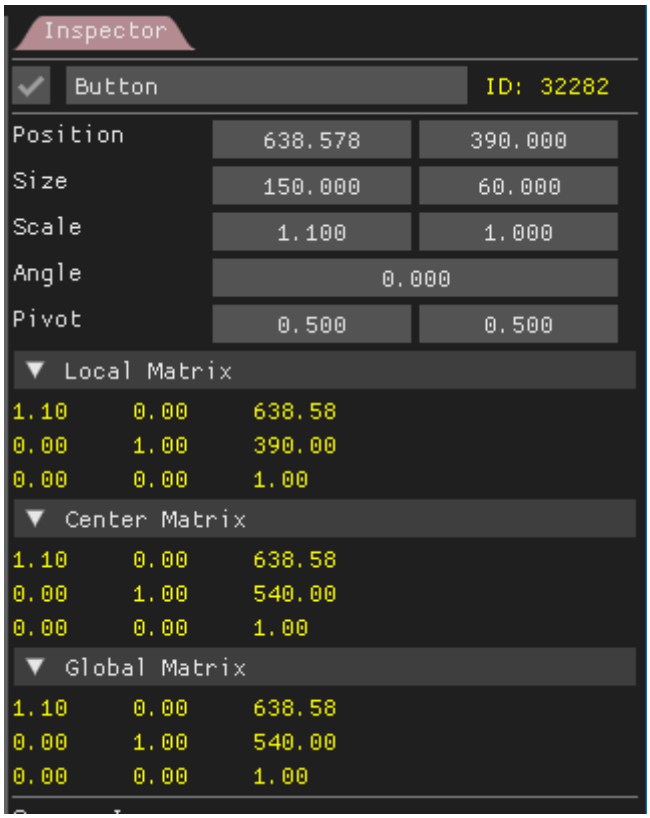
Desplega un llistat amb tot els tipus de widgets (llistats a l'apartat 6.4) per a afegir-ne un a l'escena.

Project (Projecte)

Actualment només compta amb una opció: Run Simulation (Executar Simulació). Aquesta opció inicia una simulació dels widgets existents a l'escena com si s'executés des d'un videojoc.

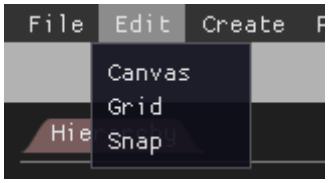
Development Tools (Eines de Desenvolupament)

Permet habilitar informació addicional sobre la llibreria per a comprovar el seu correcte funcionament. En la Figura 7-3 es pot veure aquesta informació habilitada, mostrada en color groc. Són dades que l'usuari del programa no necessita saber, però útils per qui el desenvolupa.



[Fig. 7-3] Eines de Desenvolupament

Per tal d'implementar la barra d'eines a través d'ImGui cal cridar dues funcions, primer: ImGui::BeginMainMenuBar() i ImGui::EndMainMenuBar(). Totes les funcions que es cridin entre aquestes dues s'afegiran a la barra d'eines. Per afegir nous botons a la barra s'utilitza la funció ImGui::BeginMenu(). Aquesta retorna "true" si el menú està desplegat, i també necessita ImGui::EndMenu() al final. Per a afegir noves opcions dins de cada menú, s'utilitza la funció ImGui::MenuItem(), que retorna "true" en cas que l'usuari hagi seleccionat aquesta opció. En la figura 7-4 es pot veure el resultat del codi corresponent a la figura 7-5.



[Fig. 7-4] Exemple de la barra d'eines

```
if (ImGui::BeginMainMenuBar())
{
    if (ImGui::BeginMenu("File")) { ... }

    if (ImGui::BeginMenu("Edit"))
    {
        if (ImGui::MenuItem("Canvas"))
        {
            canvas_win = true;
        }
        if (ImGui::MenuItem("Grid"))
        {
            grid_win = true;
        }
        if (ImGui::MenuItem("Snap"))
        {
            snap_win = true;
        }
        ImGui::EndMenu();
    }

    if (ImGui::BeginMenu("Create")) { ... }
    if (ImGui::BeginMenu("Project")) { ... }
    if (ImGui::BeginMenu("Dev Tools")) { ... }
    ImGui::EndMainMenuBar();
```

[Fig. 7-5] Codi de la barra d'eines

7.3 Jerarquia

El panell de jerarquia permet veure un llistat de tots els widgets col·locats en l'escena, així com la seva relació de pare-fills. Els widgets que tenen un o més fills tenen una petita fletxa a la dreta que, en prémer-la, permet veure o amagar el llistat de fills. En la Figura 7-6 es pot veure un exemple del panell de jerarquia, en què cada fill es troba més situat a la dreta.

[Fig. 7-6] Panell de jerarquia

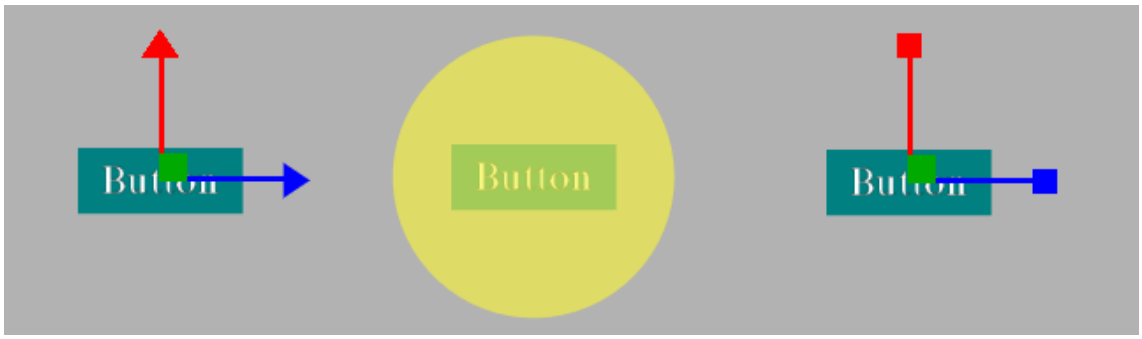
A través d'aquest panell també es poden seleccionar widgets, esborrar-los i canviar-los de pare.

Per a implementar aquest tipus de dades en ImGui es fa servir el que s'anomena tree node (node d'arbre), ja que aquest tipus de jerarquia s'anomena jerarquia en arbre. La funció necessària és ImGui::TreeNodeEx(), que retorna "true" en cas que el node tingui la fletxa desplegada i, per tant, s'ha de cridar aquesta funció de manera recursiva a través de la jerarquia.

7.4 Escena

El panell d'escena és on es mostren tots els widgets tal i com es veuran durant l'execució del videojoc i la seva distribució per la pantalla. Els widgets es poden seleccionar prement a sobre seu, i es poden moure, rotar i escalar mitjançant el que s'anomena giny (gizmos en anglès). Utilitzant les tecles W (moure), E (rotar) i R (escalar) es pot canviar

el tipus de giny, així com es pot desactivar la visualització de la graella amb la tecla G. En la Figura 7-7 es poden veure els ginys de moure, rotar i escalar, en aquest ordre.

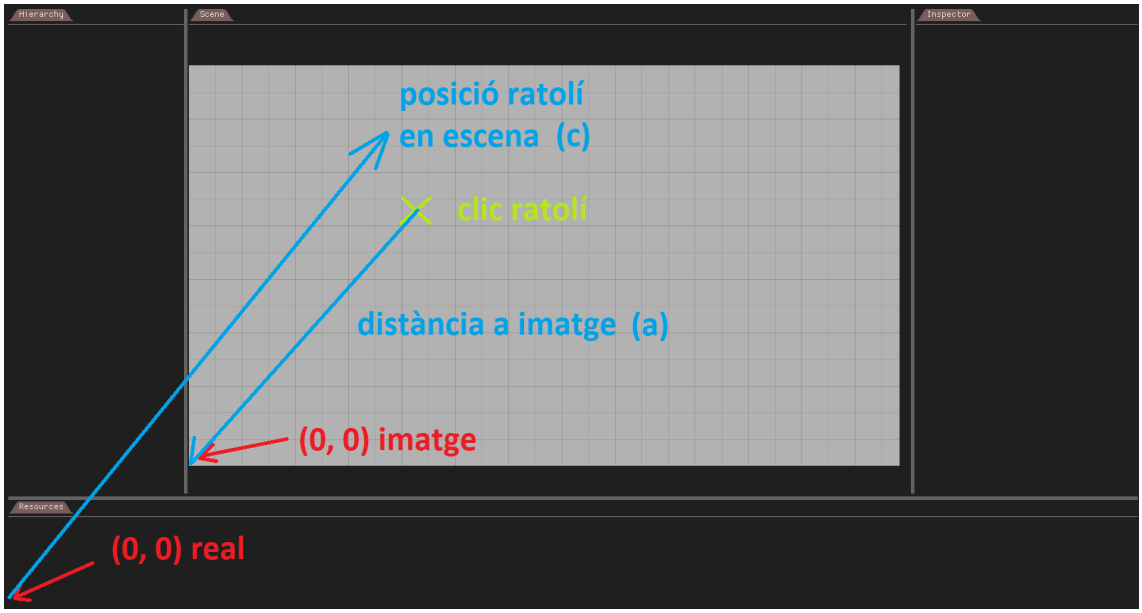


[Fig. 7-7] Ginys de moviment, rotació i escala

Per tal de tenir el panell d'escena es van haver d'implementar un seguit d'elements. Primerament, per poder mostrar una imatge per pantalla a través d'ImGui és necessari tenir les dades de la imatge guardades en un buffer de textura d'OpenGL. Donat que aquesta imatge es genera a partir del renderitzat dels widgets de la llibreria, és necessari l'ús del que s'anomena un FrameBuffer. En altres paraules, en comptes de mostrar la imatge dels widgets per la pantalla, es guarden les dades dels píxels en una textura. Una vegada generada la textura es crida la funció ImGui::Image() que la mostra per pantalla.

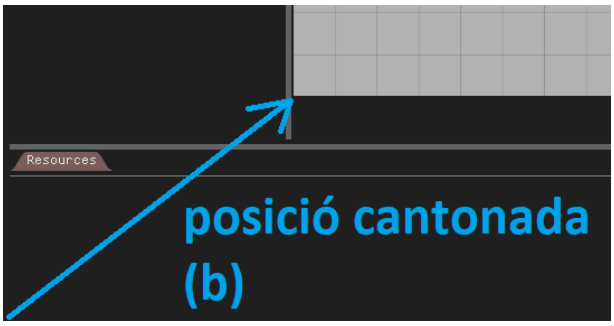
Usar aquest sistema comporta un nou problema: la mida de la textura per pantalla i la mida de l'escena real no és la mateixa, es a dir, s'agafa l'escena real de la interfície i se'n mostra una versió més petita. Això comporta que l'usuari de l'editor, en fer clics als widgets, ho faci mirant aquesta textura més petita quan, en realitat, no n'és la mida real. Per tant, quan l'usuari fa un clic en la imatge petita, cal fer la conversió en la escena de mida real.

A continuació, i utilitzant la Figura 7-8 com a referència, s'explica com dur a terme aquesta conversió.



[Fig. 7-8] Conversió clic escena I

En aquest cas hipotètic l'usuari ha fet clic a la creu de color verd. Primer de tot, cal trobar la distància respecte a la imatge, el valor A, que és un vector (x, y). Per a trobar aquest valor cal saber, també, a quin punt de la pantalla es troba la cantonada inferior esquerra de la imatge d'escena, que es pot veure en la Figura 7-9.



[Fig. 7-9] Conversió clic escena II

Aquest valor B ja és conegut. Per trobar l'A, doncs, s'ha de fer la resta entre la posició del ratolí i el valor B. Per a trobar la posició del clic respecte a l'escena real només cal multiplicar aquest vector A per la relació de mides entre la pantalla real i la imatge petita. Per tant:

C = A * (mida pantalla real / mida imatge)

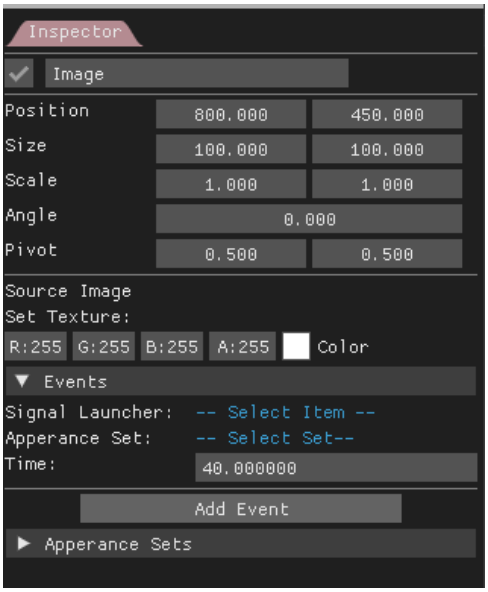
Aquest punt C és, doncs, el que s'utilitza per fer les comprovacions de la interacció de l'usuari amb els widgets, a través de la imatge petita.

Pel que fa als ginys, també serà necessaria la conversió vista anteriorment. Per a explicar com es resol aquest sistema, es tindrà en compte la posició del widget, però és aplicable de la mateixa manera a la rotació i l'escala.

Primerament, en quant l'usuari comença a utilitzar el giny, es guarda la posició del widget abans de l'ús d'aquest i la posició del clic del ratolí. En el moment en què l'usuari mou el ratolí, mantenint-lo premut, es calcula la distància entre el clic inicial i la posició del ratolí actual (utilitzant, prèviament, la conversió explicada abans). Finalment, aquesta distància se suma a la posició inicial del widget.

7.5 Inspector

El panell Inspector permet visualitzar i editar totes les dades del widget seleccionat. El nom, les seves característiques, els events i els conjunts d'aparença. En aquest apartat no s'explica en detall la implementació d'aquest panell, ja que la majoria d'usos d'ImGui són textos per a mostrar informació i elements d'entrada de dades, com InputFloat o ColorPicker. A la Figura 7-10 es pot veure el panell d'Inspector d'una UI Image.



[Fig. 7-10] Panell d'Inspector

7.5 Recursos

El panell de recursos és actualment un panell sense cap funcionalitat. S'ha posat com a part de l'editor de manera que en un futur es pugui implementar. La funció d'aquest panell és la de poder gestionar els fitxers utilitzats en el disseny de la interfície d'usuari, com les imatges o els fitxers d'escena, de manera que es puguin crear noves carpetes i es puguin ordenar dins del propi editor.

8. Conclusions i treballs futurs

Finalment, doncs, s'ha pogut veure que en poc mesos ha sigut possible portar el desenvolupament d'aquest programa de manera satisfactòria. Mitjançant aquesta eina és possible dissenyar una interfície d'usuari i, a través del fitxer generat, implementar-la en un videojoc. Així doncs, s'ha assolit l'objectiu principal d'aquest projecte. S'ha aconseguit un prototip funcional que permet veure que el desenvolupament d'un programa portat a major escala, amb un equip més gran i durant més temps, seria força útil per al sector dels videojocs. També s'ha aconseguit complir amb la resta d'objectius proposats i s'ha pogut dur a terme el projecte seguint la planificació inicial sense inconvenients.

Aquest petit prototip fa visible la necessitat d'un programa d'aquest estil per als videojocs, ja que redueix el cost en temps i recursos d'aquells equips que l'utilitzin per als seus projectes. Així doncs, la intenció per a aquest programa de UI és la de mantenir el seu desenvolupament i començar a fer-lo visible a la resta de desenvolupadors. D'aquesta manera, en el cas que la gent es comenci a interessar pel projecte i en vegi la seva utilitat, es podria buscar un equip encarat a aconseguir un producte que satisfaci les necessitats actuals. D'altra banda, el treball també pot ser útil per a aquells equips ja existents que vulguin començar a desenvolupar un programa d'aquestes característiques.

9. Bibliografia web

B1: Allotjament del projecte a Github
https://github.com/markitus18/ThorUI

B2: Versions del projecte
https://github.com/markitus18/ThorUI/releases

B3: Cplusplus – Bases de C++
http://www.cplusplus.com/doc/tutorial/classes/

B4: Autodesk retira Stingray
https://www.autodesk.com/products/stingray/overview

B5: Vídeo: principals funcionalitats de Scaleform
https://www.youtube.com/watch?v=Ot469EwnGyQ&t

B6: Qt pàgina oficial
https://www.qt.io/

B7: Projecte de Github de ImGui
https://github.com/ocornut/imgui

B8: Extensions per a ImGui
https://github.com/ocornut/imgui/wiki

B9: Unreal Engine UI
https://docs.unrealengine.com/latest/INT/Engine/UMG/

B10: Unity
https://unity3d.com/es

B11: Trello
https://trello.com/

B12: Apartat d'OpenGL a Khronos Group
https://www.khronos.org/opengl/

B13:Tutorials d'OpenGL
https://learnopengl.com/

B14: Pàgina oficial d'OpenGL
https://www.opengl.org/

B15: Pàgina oficial d'SDL
https://www.libsdl.org

B16: Tutorials d'SDL
lazyfoo.net/tutorials/SD

B17: Documentació SDL_GetKeyboardState
https://wiki.libsdl.org/SDL_GetKeyboardState

B18: Documentació SDL_PollEvent
https://wiki.libsdl.org/SDL_PollEvent

B19: Documentació SDL_GetMouseState
https://wiki.libsdl.org/SDL_GetMouseState

B20: Llibreria Parson
https://github.com/kgabis/parson

B21: Pàgina oficial JSON
https://www.json.org/

B22: Documentació Lambda C++
http://en.cppreference.com/w/cpp/language/lambda